VISIT…

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 1. Introduction: The Need for ASP.NET

IN THIS CHAPTER

-
- Problems with ASP Today
-
-
- Introducing ASP.NET
-

Before delving into the particulars of developing with C#, it will be useful to overview ASP.NET. This chapter summarizes ASP.NET's features, including some insight into how they represent improvements over ASP.old.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Problems with ASP Today

When Active Server Pages (ASP) was first introduced almost five years ago, it was seen as an answer to the awkward techniques used at that time for creating dynamic content on the Web. At the time Common Gateway Interface programs or proprietary server plug-ins were the way that most of the Web's dynamic content was created. With the release of ASP 1.0, Microsoft changed all that. ASP 1.0 provided a flexible robust scripting architecture that enabled developers to rapidly create dynamic Web applications. Developers could write in VBScript or JScript and Microsoft provided a number of services to make development easy. At the time, it was just what developers needed. As Web development matured several shortcomings of the platform became evident, and persist until today.

Separation of Code and Design

As the Web grew in popularity in the early 90s, developers experienced three distinct waves of development paradigms. In the first wave, Web developers created static HTML documents and linked them together. This was the era of the "brochure" Web site and was more about looks than anything else. The second wave brought the concept of dynamic content to the fore. Developers started creating registration forms and various small pieces of functionality and adding them into existing Web sites. The third wave was when the first and second waves came together. Web sites were being designed from the ground up to be interactive; they were treated more like an application and less like a magazine with a subscription card in it. In most instances this type of interactive page design created a development paradigm that went like so:

-
- Designers created page mockups in HTML.
-
-
- Developers added code to the pages.
-
-
- When designers needed to change their design, they copied and pasted the existing code into the new page, butchering it and destroying its functionality.
-

The severity of this problem typically depended on the size of the site, the smarts of the designers, and the techniques that developers used to guard against this mangling.

With the release of Visual Studio 6 in September 1998, it was clear that Microsoft recognized this burgeoning problem and attempted to resolve it with a new feature in Visual Basic 6, Web Classes. Web Classes made an attempt to separate the design of a page from the code that interacted with it. It enabled this separation by using an HTML template and providing a facility for doing tag replacement in the template. There were a number of problems with Web Classes. Although a great idea, they suffered from two main issues. First, the Web Classes were implemented entirely in Visual Basic, which required traditional ASP developers to shift their thinking patterns for creating applications. Second, Microsoft had scalability issues related to the threading models of ASP and Visual Basic. Because of the previously stated reasons and many other smaller ones, Web Classes never really gained any

This document is created with the unregistered version of CHM2PDF Pilot

traction among developers.

Scripting Language Based

When ASP 1.0 was first released, the fact that all development was done using scripting languages was a big plus. It meant that developers didn't have to go through a painful restart/compile process that they might have been accustomed to with CGI or ISAPI style applications. As applications grew larger, numbers of users increased and developers were using ASP for increasingly difficult problems. The fact that all code was interpreted became a potential performance bottleneck. When using VBScript there was limited support for error handling. Many developers sidestepped this issue by moving code into compiled COM objects. While this move solved some of the performance problems it created new ones in deployment and scalability.

State Management

One of the most frustrating aspects that new Web developers faced early was dealing with the stateless nature of Web development. With ASP 1.0, Microsoft introduced the concept of a Session object, which was designed to make associating state with a particular user easy. This addition was arguably one of the most compelling features of ASP 1.0. Scalability and reliability started to become important as developers began creating larger applications. To address this need, developers started deploying their applications to Web farms. Web farms use multiple servers and spread the request for pages across them somewhat equally. This makes for a great scalability story unless the developer is using that cool Session object. This object is specific to a particular machine in a Web farm and will not work if a user gets bounced to another server. So, an application that was deployed to a Web farm could not use the Session object.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Introducing ASP.NET

ASP.NET is Microsoft's answer to the aforementioned problems and many others that were not explicitly stated. It is a fundamental rewrite of ASP that has been in process for more than two years. The ASP team took a close look at the problems facing Web developers and created a brand new platform in the spirit of traditional ASP to solve those problems. Having used ASP.NET for a considerable amount of time, we can conclusively say they hit a home run with this release.

Platform Architecture

ASP.old was an Internet Server Application Programming Interface (ISAPI) filter that was written specifically to interact with Internet Information Server (IIS). It was monolithic in nature and relied very little on external services.

NOTE

Note: In the IIS 5.0 time frame, ASP did use Microsoft Transaction Server (MTS) as an external service.

ASP.NET is still an ISAPI filter. However, unlike ASP.old, ASP.NET relies on a large number of "external" services—the .NET framework. ASP.NET and the .NET framework are so tightly coupled that it is difficult to consider the .NET framework as an external service. However, since it is accessible from applications outside the scope of ASP.NET, it should be considered an "external" service. As it turns out, this is a huge win for the ASP.NET developer. No longer must the developer write everything from scratch. Instead, the .NET framework provides a large library of prewritten functionality.

The .NET framework redistributable consists of three main parts: the Common Language Runtime, the .NET framework base classes, and ASP.NET.

Common Language Runtime

The Common Language Runtime (CLR) is the execution engine for .NET framework applications. However, despite the common misconception, it is not an interpreter. .NET applications are fully compiled applications that use the CLR to provide a number of services at execution. These services include:

-
- Code management (loading and execution)
-
-
- Application memory isolation
-

This document is created with the unregistered version of CHM2PDF Pilot

- Verification of type safety
- Conversion of IL to native code
- Access to metadata
- Garbage collection
- Enforcement of code access security
- Exception handling
- Interoperability
- Automation of object layout
- Support for debugging and profiling

The CLR is a platform that abstracts functionality from the operating system. In this sense, code written to target the CLR is "platform independent" provided that there is an implementation of the CLR on the destination platform.

Managed Execution

The CLR isn't just a library or framework of functions that an executing program can call upon. It interacts with running code on a number of levels. The loader provided by the CLR performs validation, security checks, and a number of other tasks each time a piece of code is loaded. Memory allocation and access are also controlled by the CLR. When you hear about "Managed Execution," this is what folks are speaking about: the interaction between the CLR and the executing code to produce reliable applications.

Cross-Language Interoperability

This document is created with the unregistered version of CHM2PDF Pilot

One of the most frustrating things with current COM- or API-based development practices are that interfaces are usually written with a particular language consumer in mind. When writing a component to be consumed by a Visual Basic program, a developer will typically create the interfaces in a different fashion than if the component were intended to be consumed by a C++ program. This means that to reach both audiences, the developer must either use a least common denominator approach to developing the interface or must develop an interface for each consumer. This is clearly not the most productive way to write components. A second problem that most developers merely accept as normal today is that most components need to be written in a single language. If you create a component in C++ that exposes an employee object, you can't then inherit from that object in Visual Basic to create a Developer object. This means that typically a single language is chosen for most development projects to enable reuse.

.NET changes all this. Cross-language interoperability was built in from the start. All .NET languages must adhere to the Common Language Specification (CLS) that specifies the base level of functionality that each language must implement to play well with others. The CLS is written in such a way that each language can keep its unique flavor but still operate correctly with other languages within the CLR. The CLS includes a number of data types that all conforming languages must support. This restriction works to eliminate a common problem for developers: creating an interface that utilizes data types that another language doesn't support. It also supports both Binary as well as Source code inheritance, enabling the developer to create an Employee object in C# and inherit from it in Visual Basic.

What this means to you as a developer is that code reuse has become much simpler. As long as the code was written for .NET, you don't need to worry what language it was written in. In fact, the choice of language becomes more of a lifestyle choice instead of a capability choice. All languages in .NET are theoretically created equal, so you gain no performance or functionality benefit by using Visual Basic instead of C#. Use the language in which you are the most productive.

New Features in ASP.NET

Up to this point all the features mentioned are gained due to the hosting of ASP.NET on top of the .NET framework. However, these features are just the tip of the iceberg. As I mentioned previously, ASP.NET is a total rewrite, with significant features aside from the intrinsic .NET ones. We are going to give you an overview of the new features in ASP.NET while showing how these features address the problems of separation of code and design, scripting languages, and state management.

Web Forms

Web forms are Microsoft's attempt to solve the problem of the separation of code and design. ASP.NET now offers a code-behind model reminiscent of the forms designer in Visual Basic. This means that you can now place your code in a separate file and still interact with the page. This separation is done by providing a new event-driven model on top of page execution, as well as providing an object model on top of the HTML in the page. Instead of a top-to-bottom linear execution model, events are raised during the execution of a page. Your code sinks those events and responds to them by interacting with the object model that sits on top of the HTML.

This quite neatly solves the issue of designers modifying the HTML and breaking code.

In addition to the new execution model, Microsoft has also created a new server-side control model. Unlike the lame Design Time Controls in Visual Interdev, these new models are incredibly useful encapsulations of common display

This document is created with the unregistered version of CHM2PDF Pilot

paradigms. They do not introduce any browser dependencies and they run on the server, not the client. In the few cases where they emit browser-dependent code, they sniff the browser and degrade gracefully. More information on Web forms can be found in Chapter 2, "Page Framework."

Web Services

As we move beyond the first and second generations of Web applications, it has become apparent that the paradigm of the Web can be extended to solve problems that predate it. In the past, businesses exchanged information using Electronic Data Interchange (EDI) over Value Added Networks (VANs). This worked well enough, but the cost of gaining access to a VAN as well as the complexity of implementing various industry-specific EDI protocols excluded all but the largest companies from participating in the exchange of information.

Web services are a way to transfer the same types of information over the Internet (instead of expensive VANs) using industry-standard protocols such as HTTP, XML, and TCP/IP. Web services are now so easy to create in .NET that individuals or businesses of any size should be able to play in this space. Web services aren't limited to replacing traditional EDI protocols. They open up many opportunities that EDI has never made inroads into. Jump ahead to Chapter 6, "Web Services," for more information.

Data Access

When ASP 1.0 first shipped, the data access story at Microsoft was in a state of flux. At the time, Remote Data Objects (RDO) was the technology of choice for Visual Basic developers. ActiveX Data Objects (ADO) was introduced with the shipment of Visual Basic 5.0 in February 1997. It was intended to be a new data access model for all types of data and was paired with another new technology, OLE DB.

While ADO was great for what it was designed for—connected data access—it fell short in the disconnected arena. Features were added in successive versions to allow it to work in a disconnected fashion. ADO 1.0 had no support for XML. ADO 1.0 could not predict the success of XML as a data description language when it was shipped, and XML support was cobbled onto later versions. Neither of these features were designed in from the beginning.

ADO.NET is a new data access technology taking advantage of all the things Microsoft learned with ADO, RDO, OLEDB, ODBC, and other preceding data access implementations. It was designed from the beginning to be coupled very tightly to XML and work effectively in a disconnected fashion. For more information see Chapter 11, "Creating Database Applications with ADO.NET."

Deployment

One of the perennial arguments among ASP developers was how much code to migrate into COM objects. Some writers advocated all code living in COM objects and ASP should only contain a single-method call to invoke the COM object. While this might have been great in theory, it eliminated one of the biggest strengths of ASP: the capability to rapidly create an application and make changes on-the-fly. With all the logic and HTML tied up in COM objects, a simple HTML tag change meant recompiling and redeploying the COM objects. The biggest problem in our minds lies with using this approach. COM objects are Dynamic Link Libraries (DLL) that are dynamically loaded by IIS. While loaded they cannot be replaced. To deploy a COM object the developer needed to shut down IIS, shut down the MTS packages the COM object lived in, replace it, and then restart IIS. This summary is actually a simplification of the process, but you can see the problems this technique brings to the fore.

This document is created with the unregistered version of CHM2PDF Pilot

Each time a new version is deployed, the Web server must go down! The downtime this technique causes can be handled by creating Web farms and doing rolling upgrades; however, in a large Web farm this means a simple change can take literally hours to deploy as the new objects are rolled out.

With the code-behind model inherent in ASP.NET, this situation could have been exacerbated. Instead, Microsoft vastly simplified the deployment model. Comp-onents, now called assemblies, no longer require registration on a machine for deployment. Assemblies are the .NET equivalent of a COM object. They are self describing and contain a manifest which contains metadata about the assembly. The metadata includes things like the version, the assemblies it depends on, and potentially, its security identity.

Deployment is as easy as copying the assemblies into a /bin directory in the application root. ASP.NET will notice that a new version has been copied over and unload the old version and load the new version! Deployment becomes as simple as an XCOPY /S or a recursive FTP to upload the new files to the Web server. For more information see Chapter 5, "Configuration and Deployment."

Configuration

In the past, all configuration information for ASP was stored as part of the IIS metabase. This was a binary file analogous to the registry that held all settings for IIS and ASP. The only ways to effect changes were to use the Internet Services Manager or to write scripts that utilized the Active Directory Services Interfaces (ADSI) to automate the changes. This process made it very difficult to synchronize the settings of multiple servers in a Web farm.

ASP.NET introduces a new paradigm for all settings. Instead of being stored in the opaque metabase, they are now stored as a hierarchical set of XML configuration files. These files live in the application root and subdirectories. So, now as a developer uses XCOPY to deploy source files, the settings are also deployed! No need to write a bunch of configuration scripts anymore. For more information see Chapter 5.

State Management

State management has been vastly improved in ASP.NET. Now, three options exist for maintaining state on the server. The classic ASP 3.0 method of in-memory state on a single server still exists. In addition, an out-of-process state server and a durable state option is stored in SQL Server.

The other limitation of state services in ASP.old was the reliance on cookies for connecting a user back up to their state. ASP.NET introduces a new option for cookieless state that performs URL munging to connect a user to state information. For more information see Chapter 4, "State Management and Caching."

Caching

The reason most developers use ASP is to lend a dynamic nature to the Web. This could mean accessing backend databases for data or perhaps pulling it in from nontraditional backends. The problem with this dynamic content is that while one can easily scale the Web tier using a scale-out methodology of multiple Web servers, this scaling is not as easily done in the data tier. Scale-out approaches for databases are just beginning to appear. Until these approaches are perfected, how can Web developers scale applications?

This document is created with the unregistered version of CHM2PDF Pilot

For data that changes infrequently, caching is a great solution. ASP.NET offers two forms of caching. Output caching takes an entire page and stores the executed results in memory for later delivery. Data caching takes items that were expensive to create, such as DataSets, and caches them on the server side. For more information see Chapter 4.

Debugging

Debugging ASP applications has always been difficult. While remote debugging solutions were offered in previous versions of Visual Studio, precious few developers were able to get them to work consistently. Consequently, most debugging consisted of Response.Write statements littered throughout code or some type of logging mechanism that the developer created.

ASP.NET not only improves remote debugging and makes it consistent, it also offers a new Trace facility that is great for handling the types of things that logging or Response.Write were used for in the past. For more information see Chapter 3, "Debugging ASP.NET Applications."

Availability

Anybody that currently has a site on the Web knows that availability is key. If your site is down, a visitor can turn to a million others. Problem is, they might not come back!

ASP.NET has introduced a number of process controls that are aimed directly at improving availability. Now the ASP.NET process can restart automatically based on things such as memory utilization, time up, or even number of requests handled, which helps cope with situations where ASP used to get jammed up. For more information see Chapter 5.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 2. Page Framework

IN THIS CHAPTER

- ASP.NET's Control Model
- Separating Presentation from Code Using Code Behind
- Programming HTML Controls
- Attributes of the Page Object
- Creating User Interfaces with Web Controls
- Server Controls and Page Object Reference

Programming an ASP.NET application is significantly different than programming in ASP.old. The difference can be likened to the change that occurred when moving from QuickBasic programming to Visual Basic programming.

The changes in ASP.NET can be broken down into three categories: the control model, the event model, and the separation of code from presentation.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

ASP.NET's Control Model

In QuickBasic, your code dealt with the screen as a long piece of paper to which you sent output. You may have used screen libraries that encapsulated a lot of this functionality and made it a much higher-level operation with better positioning.

With the advent of Visual Basic, you moved into a world of reusable controls. You designed your UI by dragging and dropping controls onto a design surface instead of outputting text to the screen or drawing polygons.

These controls were objects that had methods and properties that were used to customize their display and functionality. ASP.old was in many ways similar to QuickBasic. The entire page is essentially treated as a long piece of paper onto which your code placed content. No object model gives you access to the HTML that surrounds your code—just a way for you to output additional HTML based on the location of your code.

ASP.NET changes this by introducing the concept of server controls. If you used Visual Interdev to create ASP.old Web applications, you may be thinking, "Great! They just renamed those onerous design-time controls!" This is not the case. Server controls are not design-time controls in another guise. Nor do server controls require any particular type of client—in other words, server controls aren't ActiveX Controls or client-side behaviors. Server controls are a high-level abstraction of functionality utilized during page execution to place user-interface elements onto the page.

Let's take a look at this. Listing 2.1 shows the HTML for a traditional ASP.old form.

Listing 2.1 A Simple ASP.old Page, SimplePage.asp

```
<html>
<head>
    <title>SimplePage.asp</title>
</head>


<body>
    <form name="WebForm1" method="post">
        <p>
        <table border=0>
            <tr>
                <td>Name:</td>
                <td><input type=text name=txtName></td>
                <td><input type=submit name=Button1 Value="Send"></td>
            </tr>
            <tr>
                <td valign=top>Hobby:</td>
                <td>
                    <select name=lbHobbies Multiple>
                        <option Value="Ski">Ski</option>
                        <option Value="Bike">Bike</option>
                        <option Value="Swim">Swim</option>
                    </select>
                </td>
                <td> </td>
            </tr>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        </table>
        </p>
    </form>
</body>
</html>
```

What happens when a user fills in a name, chooses a hobby, and presses the Send button? The page is first posted back to the server. No code is in the form at this point, so all the selections that the user made in the Select tag (information that we'll refer to as form state) are lost. The page is then returned back to the browser. In ASP.old, if you want to preserve the form state, you are forced to write code to do that.

Listing 2.2 contains SimplePage2.asp showing the typical code you would write with ASP.old to make this work.

Listing 2.2 SimplePage2.asp Showing Code to Preserve Form State in ASP.OLD

```
<html>
<head>
    <title>SimplePage2.asp</title>
</head>

<SCRIPT LANGUAGE="VBScript" RUNAT=SERVER>
    function IsOptionSelected(strControlName, strOption)
        for iCount = 1 to Request(strControlName).Count
            if request(strControlName)(iCount) = strOption then
                response.write " SELECTED "
            end if
        next
    end function

</SCRIPT>

<body>
    <form name="WebForm1" method="post">
        <p>
        <table border=0>
            <tr>
                <td>Name:</td>
                <td><input type=text name=txtName value="<% = Request("txtName")
%>"></td>
                <td><input type=submit name=Button1 Value="Send"></td>
            </tr>
            <tr>
                <td valign=top>Hobby:</td>
                <td>
                    <select name=lbHobbies Multiple>
                        <option <% IsOptionSelected "lbHobbies", "Ski" %>
Value="Ski">Ski</option>
                        <option <% IsOptionSelected "lbHobbies", "Bike" %>
Value="Bike">Bike</option>
                        <option <% IsOptionSelected "lbHobbies", "Swim" %>
Value="Swim">Swim</option>
                    </select>
                </td>
                <td> </td>
            </tr>
        </table>
        </p>
    </form>
</body>
</html>
```

This document is created with the unregistered version of CHM2PDF Pilot

With the advent of server controls, ASP.NET adds functionality to HTML's own user-interface controls, making them do what you would expect them to do; that is, save the data that the user just spent time typing in.

You need to do three things to make ASP.NET server controls work.

1.

1. ASP.NET server controls are identified using the ID attribute instead of (or in addition to) the Name attribute. You are allowed to use both. You may want to use the Name attribute if you have client-side script that needs to refer to the control.
1.
2.

2. ASP.NET server controls require you to add the runat=server attribute. This attribute indicates to ASP.NET that the tag is something more than a built-in HTML tag.
2.
3.

3. ASP.NET server controls require a closing tag. Server controls are implemented using XML namespaces and, like XML, require every element to have a matching closing element. You can use XML style syntax as a shortcut creating a tag such as <input type=text runat=server />.
3.

So let's do this to the code that was in Listing 2.1. Listing 2.3 shows simplepage.aspx, an ASP.NET implementation of simplepage.asp.

Listing 2.3 SimplePage.aspx—A Reworking of Listing 2.1 in ASP.NET

```
<html>
<head>
    <title>SimplePage.aspx</title>
</head>

<body>
    <form id="WebForm1" method="post" runat="server">
        <p>
        <table border=0>
            <tr>
                <td>Name:</td>
                <td><input type=text id=txtName runat=server /></td>
                <td><input type=submit id=Button1 Value="Send" runat=server /></td>
            </tr>
            <tr>
                <td valign=top>Hobby:</td>
                <td>
                    <select id=lbHobbies Multiple runat=server>
                        <option Value="Ski">Ski</option>
                        <option Value="Bike">Bike</option>
                        <option Value="Swim">Swim</option>
                    </select>
                </td>
                <td> </td>
            </tr>
        </table>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        </p>
    </form>
</body>
</html>
```

All that's changed is the addition of the runat=server attribute to the form tag, the input tag, and the select tag. We've also changed each of the name attributes to ID attributes. That's it. If you run this page, fill in a name, select a hobby, and then click the Send button. The data that you entered stays there after the page is destroyed and re-created on its round trip to the server. The server controls realize that the desired default behavior is to maintain input; that is, they maintain their state, and they do so automatically.

If you don't want a given server control to maintain its state, you can use a new attribute with any server control called EnableViewState. By setting this to false, you can override the default behavior of maintaining form state across posts.

Two categories of server controls are HTML controls and Web controls. The HTML controls mirror their HTML counterparts. HTML controls include the following:

HTML Control Class	**HTML Tag**
HtmlAnchor	Anchor
HtmlButton	<button />
HtmlContainerControl	Any control that requires a closing tag
HtmlControl	Any HTML server control
HtmlForm	<form></form>
HtmlGenericControl	Represents any HTML tag without a specific server control class (for example, <p>)
HtmlImage	
HtmlInputButton	<input type=Button />
HtmlInputCheckBox	<input type=Checkbox />
HtmlInputControl	Any <input type=* /> control
HtmlInputFile	<input type=file />
HtmlInputHidden	<input type=hidden />
HtmlInputImage	<input type=image />
HtmlInputRadioButton	<input type=Radio />
HtmlInputText	<input type=Text />
HtmlSelect	<select> </select>
HtmlTable	<table> </table>
HtmlTableCell	<td> </td>
HtmlTableCellCollection	All <TD> or <TH> tags within <table> </table>
HtmlTableRow	<tr> </tr>
HtmlTableRowCollection	All <TR> tags within <table> </table>
HtmlTextArea	<textarea> </textarea>

NOTE

This document is created with the unregistered version of CHM2PDF Pilot

All these tags require the runat=server attribute to make them HTML controls. If you forget to add this attribute, these controls will be treated as normal HTML tags. They will be programmable only via client-side code, which may not be what you want.

These controls wrap the related HTML tag with a complete object model that allows access to all the attributes of the tag via properties or methods. You see examples of this later on in this chapter.

Web controls don't always map directly to a single HTML tag. In many cases they are composite controls that represent a large number of HTML tags. Let's take a look at an example. Listing 2.4 shows the Calendar Web control.

Listing 2.4 Calendar.aspx Showing a Single Web Control

```
<html>
<head>
    <title>Calendar.aspx</title>
</head>

<body>
    <form id="WebForm1" method="post" runat="server">
        <asp:calendar id=Calendar1 runat=server />
    </form>
</body>
</html>
```

Save this file as Calendar.aspx, and that's it. But that one HTML tag generates something that looks like Figure 2.1.

Figure 2.1. The output of Calendar.aspx from Listing 2.4

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Separating Presentation from Code Using Code Behind

Code in ASP.old was often difficult to maintain because it was interspersed with HTML markup. Even when using a visual development tool such as Visual InterDev or Dreamweaver UltraDev, it could be difficult and time consuming to track down a chunk of ASP code that needed to be debugged.

The solution to this problem is a tactic that developers on many platforms typically use: separating logic (the code that you write) from presentation (the way the data appears). By separating logic from presentation, you can be assured that all the code is located in the same place, organized the way you want, and easily accessible. Separating logic from presentation also minimizes the possibility that you'll generate new bugs in the presentation code as you debug the core logic of the application.

One tactic for separating code from presentation in ASP.NET is code behind. Code behind is a feature that enables you to take most of or all the code out of an ASP.NET page and place it in a separate file. The code is processed normally; the only difference is where the code is located.

Visual Basic actually introduced the concept of code behind. The idea was that the code that dealt with a particular object was in a different layer "behind" the object. Of course, this was only one way to describe it. In reality, code behind was just a separate source file for each form that encapsulated the code related to that form. The code in the file was actually tightly coupled with the form.

ASP.NET has this same concept with a slight twist. If you write a page as shown in Listing 2.7, behind the scenes ASP.NET parses the code out of the page for you—invisibly in the background. A class is created that inherits from System.Web.Page and includes a class level object declaration for each runat=server control in your page. This is done by default to enable you to continue programming using the simple model provided in ASP.old.

Alternatively, you can create this class yourself and derive the page from it. This separates the code from the layout of the page. This separation is potentially a huge benefit. In my previous life I worked with a company whose standard development process went something like this: A business process owner would decide that some feature should be Web enabled. The owner would come to a designer with, at best, a couple of sketches of what the Web pages needed to implement this feature should look like. The owner would then work with the designer to create a series of HTML pages that represented the feature. These pages would then be handed off to a developer to "activate" them. The developer would go through the pages, adding the code to actually make the feature work. When the developer was done, the feature was then shown to the business process owner. Inevitably, the owner would realize that several features had been missed and/or additional features were needed So the process would start over again. The designer would take the completed pages and start moving the HTML around to meet the needs of the change requests. After the pages were again looking good, the designer would hand off to the developer. The developer would open up the pages and throw his or her hands up in despair. In the process of reformatting and rearranging the HTML, the designer inevitably would have scrambled the ASP.old code that had lived intermixed with the HTML. In many instances, it was easier for the developer to just rip the old code out and re-add it via copy/paste from the first version. This iterative process could continue for dozens of rounds, depending on the complexity of the feature.

This document is created with the unregistered version of CHM2PDF Pilot

I suspect my previous company and I were not the only ones frequently faced with this issue. It begs for a new model that allows the separation of the layout and formatting from the code that operates on it. ASP.NET is not Microsoft's first attempt at this concept. It was tried, as part of Web Classes in Visual Basic 6.0 but was not very successful. I predict that ASP.NET will be a much more successful implementation.

The way that code behind in ASP.NET works is that you create a class that inherits from System.Web.UI.Page. This is the base class for a page.

NOTE

A complete reference to the Page object can be found at the end of this chapter.

The .aspx page then inherits from the class you create. This inheritance is accomplished via the @Page directive that is discussed in further detail in this chapter. The @Page directive Inherits attribute enables you to indicate from which class the page should inherit.

The Src attribute enables you to indicate from which file the source code should be dynamically compiled. This last attribute is not required if the class has already been compiled and is in the Global Assembly Cache. Alternatively, under the directory the page is in, you can create a special directory called /bin. This directory is one of the first places ASP.NET looks for already compiled code. If the code has not already been compiled, the file reference by the Src attribute is compiled and looked at for the class specified in the Inherits attribute. Listing 2.8 shows the aspx page for the sample we have been looking at. Note that no code is in this page, just HTML markup.

Listing 2.8 SimplePage3.aspx Using Code Behind—This is the .aspx Page

```
<% @Page src="simplepage3.aspx.cs" Inherits="SimplePage" %>
<html>
<head>
   <title>SimplePage3.aspx</title>
</head>

<body>
    <form id="WebForm1" method="post" runat="server">
      <p>
      <table border=0>
         <tr>
            <td>Name:</td>
            <td><asp:textbox id=txtName runat=server /></td>
            <td><asp:button id=Button1 Text="Send" runat=server /></td>
         </tr>
         <tr>
            <td valign=top>Hobby:</td>
            <td>
               <select id=lbHobbies Multiple runat=server>
                  <option Value="Ski">Ski</option>
                  <option Value="Bike">Bike</option>
                  <option Value="Swim">Swim</option>
               </select>
            </td>
            <td> </td>
         </tr>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
      </table>
      </p>
      <asp:label id=lblOutput runat=server />
    </form>
</body>
</html>
```

Also note the @Page tag that indicates the code for this page is in a file called SimplePage3.aspx.cs. The class that implements the functionality for this page is called SimplePage. Listing 2.9 shows SimplePage3.aspx.cs.

Listing 2.9 Simplepage3.aspx.cs Is the Code Behind File for SimplePage3.aspx

```
public class SimplePage : System.Web.UI.Page
{
   protected System.Web.UI.WebControls.Button Button1;
   protected System.Web.UI.WebControls.TextBox txtName;
   protected System.Web.UI.WebControls.Label lblOutput;
   protected System.Web.UI.HtmlControls.HtmlSelect lbHobbies;

   private void Page_Init()
   {
      Button1.Click += new System.EventHandler(Button1_Click);
   }
   private void Button1_Click(object sender, System.EventArgs e)
   {
      string strTemp;

      // Build up the output
      strTemp = "Name:" + txtName.Text + "<BR>Hobbies: ";
      for(int iCount = 0; iCount <= lbHobbies.Items.Count - 1; iCount++)
         if(lbHobbies.Items[iCount].Selected)
            strTemp = strTemp + lbHobbies.Items[iCount].Text + ", ";

      // Place it into the label that was waiting for it
      lblOutput.Text = strTemp;
   }
}
```

This looks very similar to the previous example, except that no code is in the page! It is strictly markup. One cool feature is that by altering the Inherits attribute, you can tie more than one aspx page to the same code-behind file. If you type the code in the previous two listings into Notepad and save using the recommended filenames, you will have a working page.

Code behind gives you an additional way to wire up events. It's best not to make the HTML markup know any more than it needs to about the code. Using the approach shown earlier to wire up events, you're required to know the name of the event procedure in the code behind. An alternative is to define the page event handlers in Page_Init(). In Listing 2.9 the Button1_Click function is defined as a handler for Button1.Click event. This is an alternative way to wire up the event handlers for ASP.NET with code behind. With this technique, the HTML markup doesn't have to know anything about the code-behind class.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Programming HTML Controls

In this section we'll take a look, one by one, at the HTML controls provided by ASP.NET and show you examples of some of the more interesting ones.

HtmlAnchor

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.Dll.

The HtmlAnchor control encapsulates the <a> tag with a server-side control model. You shouldn't use this for every link, but it makes it easy to dynamically generate links as needed.

Properties		
Attributes	ClientID	Controls
Disabled	EnableViewState	Href
ID	InnerHtml	InnerText
Name	NamingContainer	Page
Parent	Site	Style
TagName	Target	TemplateSourceDirectory
Title	UniqueID	Visible
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerClick
Unload		

Listings 2.10 and 2.11 show a page that alters a link based on whether the page containing the link is accessed using HTTPS. This is a frequent requirement when building secure e-commerce Web sites.

Listing 2.10 The HTML for a Dynamically Generated Anchor Using the HtmlAnchor Control

```
<%@ Page language="c#" Codebehind="Anchor.aspx.cs" AutoEventWireup="false"
Inherits="HtmlControls.Anchor" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        <title>Dynamically Generated Anchor</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="Anchor" method="post" runat="server">
            <a id="AnchorTag" runat="server">Test Anchor</a>
        </form>
    </body>
</HTML>
```

Listing 2.11 The Code for a Dynamically Generated Anchor Using the HtmlAnchor Control

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace HtmlControls
{
    public class Anchor : System.Web.UI.Page
    {
        protected System.Web.UI.HtmlControls.HtmlAnchor AnchorTag;

        private void Page_Load(object sender, System.EventArgs e)
        {
            if(Page.Request.IsSecureConnection)
            {
                AnchorTag.HRef = "https://www.deeptraining.com";
                AnchorTag.InnerText = "Secure Link";
            }
            else
            {
                AnchorTag.HRef = "http://www.deeptraining.com";
                AnchorTag.InnerText = "Unsecure Link";
            }

        }
        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        }
        #endregion
    }
}
```

The ServerClick event enables you to optionally process the anchor on the server side instead of the client side. This adds an extra round trip to the action but allows you to treat text just like buttons that invoke server-side actions. The InnerText or InnerHtml properties enable you to alter the content between the <a> and tags, as shown in Listing 2.11. The Title property corresponds to the alt text or ToolTip that pops up for an anchor. The code in Listings 2.10 and 2.11 was generated using VS.NET as opposed to Listings 2.8 and 2.9, which used Notepad.

HtmlButton

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlButton class provides a server-side encapsulation of the HTML 4.0 <button> tag. It works only in Internet Explorer 4.0 and later. If you want to use a button that works in a wider variety of browsers, take a look at HtmlInputButton later on in this section.

Properties		
Attributes	CausesValidation	ClientID
Controls	Disabled	EnableViewState
ID	InnerHtml	InnerText
NamingContainer	Page	Parent
Site	Style	TagName
TemplateSourceDirectory	UniqueID	Visible
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerClick
Unload		

This control is primarily used to kick off some server-side processing. This is done through the ServerClick event. Listings 2.12 and 2.13 show a page with an HtmlButton control that fires off some script in the page to write some text to the page.

Listing 2.12 The HTML for Button.aspx

This document is created with the unregistered version of CHM2PDF Pilot

```
<%@ Page language="c#" Codebehind="Button.aspx.cs" AutoEventWireup="false"
Inherits="HtmlControls.Button" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>HtmlButton Class</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="Button" method="post" runat="server">
            <button id="btnClick" title="" type="button" runat="server">Click
Me</button>
        </form>
    </body>
</HTML>
```

Listing 2.13 The Code for Button.aspx

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace HtmlControls
{
    /// <summary>
    /// Summary description for Button.
    /// </summary>
    public class Button : System.Web.UI.Page
    {
        protected System.Web.UI.HtmlControls.HtmlButton btnClick;

        private void Page_Load(object sender, System.EventArgs e)
        {
            btnClick.InnerText = "Click Me!";
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            this.btnClick.ServerClick += new
System.EventHandler(this.btnClick_ServerClick);
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void btnClick_ServerClick(object sender, System.EventArgs e)
        {
            Response.Write("You clicked me!");
        }
    }
}
```

HtmlForm

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlForm class allows you to change the properties of a form on the server side. These properties include the target you are posting to as well as the method used to send the data to the server.

Properties		
Attributes	ClientID	Controls
Disabled	EnableViewState	EncType
ID	InnerHtml	InnerText
Method	Name	NamingContainer
Page	Parent	Site
Style	TagName	Target
TemplateSourceDirectory	UniqueID	Visible
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

The EncType property controls the encoding of the form. Valid encoding types include "multipart/form-data", "text/plain", and "image/jpeg". Listing 2.14 and 2.15 alter the Method property to determine how the form data is posted to the server.

NOTE

This document is created with the unregistered version of CHM2PDF Pilot

Only one <form runat=server> tag is allowed per page. You might initially think this is a limitation; however, the routing of events by ASP.NET alleviates this problem immensely.

Listing 2.14 The HTML for Form.aspx

```
<%@ Page language="c#" Codebehind="Form.aspx.cs" AutoEventWireup="false"
Inherits="HtmlControls.Form1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Dynamically Altering Form Method</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="Form" method="post" runat="server">
            <asp:RadioButtonList id="RadioButtonList1" runat="server"
AutoPostBack="True">
                <asp:ListItem Selected="True" Value="Post">Post</asp:ListItem>
                <asp:ListItem Value="Get">Get</asp:ListItem>
            </asp:RadioButtonList>
        </form>
    </body>
</HTML>
```

Listing 2.15 The Code for form.aspx

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace HtmlControls
{
    /// <summary>
    /// Summary description for Form.
    /// </summary>
    public class Form1 : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.RadioButtonList RadioButtonList1;
        protected System.Web.UI.HtmlControls.HtmlForm Form;

        private void Page_Load(object sender, System.EventArgs e)
        {
            Response.Write("Form Method used to Send: " + Form.Method);
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.RadioButtonList1.SelectedIndexChanged += new
System.EventHandler(this.RadioButtonList1_SelectedIndexChanged);
            this.ID = "Form1";
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void RadioButtonList1_SelectedIndexChanged(object sender,
System.EventArgs e)
        {
            Form.Method = RadioButtonList1.SelectedItem.Value;
        }
    }
}
```

NOTE

In the code example in Listing 2.15, you will have to press the radio button twice to see the results. This is because the procedure that outputs the form method runs prior to the RadioButtonList1_SelectedIndexChanged event that actually alters the way the form works.

HtmlImage

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlImage class encapsulates the HTML <img> tag. This can be useful for dynamically pointing to multiple images on your site. Note that this does not enable you to send the actual image data. You can alter only where the browser retrieves the image data from.

Properties		
Align	Alt	Attributes
Border	ClientID	Controls

This document is created with the unregistered version of CHM2PDF Pilot

Disabled	EnableViewState	Height
ID	NamingContainer	Page
Parent	Site	Src
Style	TagName	TemplateSourceDirectory
UniqueID	Visible	Width
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

Listings 2.16 and 2.17 use the image-specific properties of this server control to allow you to alter how an image is displayed on the page.

Listing 2.16 The HTML for image.aspx

```
<%@ Page language="c#" Codebehind="Image.aspx.cs" AutoEventWireup="false"
Inherits="HtmlControls.Image" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Image Properties</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="Image" method="post" runat="server">
            <table>
                <tr>
                    <td>
                        <img src="deeplogo2.jpg" runat="server" id="IMG1">
                    </td>
                </tr>
                <tr>
                    <td>
                        Alt:  <input type="text" id="txtAlt" runat="server"
value="Deep Logo" NAME="txtAlt">
                        <br>
                        <input type="checkbox" id="chkBorder" runat="server"
NAME="chkBorder">Border
                            <input type="checkbox" id="chkVisible"
runat="server" Checked NAME="chkVisible">Visible
                        <br>
                        Alignment: 
                        <select id="ddAlignment" runat="server" NAME="ddAlignment">
                            <option Value="left" selected>
                                Left</option>
                            <option Value="center">
                                Center</option>
                            <option Value="right">
                                Right</option>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
                            <option Value="top">
                                Top</option>
                            <option Value="middle">
                                Middle</option>
                            <option Value="bottom">
                                Bottom</option>
                        </select>
                        <br>
                        Size:  <input type="text" id="txtWidth" runat="server"
Width="51px" Height="24px" NAME="txtWidth">
                         x <input type="text" id="txtHeight" runat="server"
Width="51px" Height="24px" NAME="txtHeight">
                        <br>
                        Src: <input type="text" id="txtSrc" runat="server"
value="deeplogo2.jpg" NAME="txtSrc">
                        <P></P>
                        <P>
                        </P>
                        <P>
                            <input type="submit" id="btnApply" runat="server"
Value="Apply Settings" NAME="btnApply">
                        </P>
                    </td>
                </tr>
            </table>
        </form>
    </body>
</HTML>
```

Listing 2.17 The Code for image.aspx

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace HtmlControls
{
    /// <summary>
    /// Summary description for Image.
    /// </summary>
    public class Image : System.Web.UI.Page
    {
        protected System.Web.UI.HtmlControls.HtmlImage IMG1;
        protected System.Web.UI.HtmlControls.HtmlInputText txtAlt;
        protected System.Web.UI.HtmlControls.HtmlInputCheckBox chkBorder;
        protected System.Web.UI.HtmlControls.HtmlInputCheckBox chkVisible;
        protected System.Web.UI.HtmlControls.HtmlSelect ddAlignment;
        protected System.Web.UI.HtmlControls.HtmlInputText txtWidth;
        protected System.Web.UI.HtmlControls.HtmlInputText txtHeight;
        protected System.Web.UI.HtmlControls.HtmlInputText txtSrc;
        protected System.Web.UI.HtmlControls.HtmlInputButton btnApply;

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.btnApply.ServerClick += new
System.EventHandler(this.btnApply_ServerClick);
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void btnApply_ServerClick(object sender, System.EventArgs e)
        {
            // Set the alt text
            IMG1.Alt = txtAlt.Value;
            // If the border is checked set a border width
            if(chkBorder.Checked)
                IMG1.Border = 5;
            else
                IMG1.Border = 0;
            // Set the image alignment
            IMG1.Align = ddAlignment.Items[ddAlignment.SelectedIndex].Value;
            // If a width is entered then set it
            if(txtWidth.Value != "")
                IMG1.Width = int.Parse(txtWidth.Value);

            // If a height is entered then set it
            if(txtHeight.Value != "")
                IMG1.Height = int.Parse(txtHeight.Value);

            // Set the image to show
            IMG1.Src = txtSrc.Value;
            // Set whether it is visible
            IMG1.Visible = chkVisible.Checked;
        }
    }
}
```

HtmlInputButton

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlInputButton class wraps up several HTML tags, including <input type=button>, <input type=reset>, and <input type=submit>. This class is supported in all browsers (unlike the HtmlButton class).

This document is created with the unregistered version of CHM2PDF Pilot

Properties		
Attributes	CausesValidation	ClientID
Controls	Disabled	EnableViewState
ID	Name	NamingContainer
Page	Parent	Site
Style	TagName	TemplateSourceDirectory
Type	UniqueID	Value
Visible		
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerClick
Unload		

Listings 2.18 and 2.19 show the difference in behavior between the type=submit and type=reset buttons.

Listing 2.18 The HTML for inputbutton.aspx

```
<%@ Page language="c#" Codebehind="InputButton.aspx.cs" AutoEventWireup="false"
Inherits="HtmlControls.InputButton" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>HtmlInputButton Example</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="InputButton" method="post" runat="server">
            <asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
            <input type="submit" id="btnSubmit" runat="server" value="Submit"
NAME="btnSubmit">
            <input type="reset" id="btnReset" runat="server" value="Reset"
NAME="btnReset">
        </form>
    </body>
</HTML>
```

Listing 2.19 The Code for inputbutton.aspx

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace HtmlControls
{
    /// <summary>
    /// Summary description for InputButton.
    /// </summary>
    public class InputButton : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.TextBox TextBox1;
        protected System.Web.UI.HtmlControls.HtmlInputButton btnSubmit;
        protected System.Web.UI.HtmlControls.HtmlInputButton btnReset;

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.btnSubmit.ServerClick += new
System.EventHandler(this.btnSubmit_ServerClick);
            this.btnReset.ServerClick += new
System.EventHandler(this.btnReset_ServerClick);
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void btnSubmit_ServerClick(object sender, System.EventArgs e)
        {
            Response.Write("You clicked submit.");
        }

        private void btnReset_ServerClick(object sender, System.EventArgs e)
        {
            Response.Write("You clicked reset.");
        }
    }
}
```

HtmlInputCheckBox

Member of System.Web.UI.HtmlControls.

This document is created with the unregistered version of CHM2PDF Pilot

Assembly: System.Web.dll.

HtmlInputCheckBox encapsulates the <input type=checkbox> tag. The Checked property indicates whether the item was checked. Listing 2.17 uses the HtmlInputCheckbox to indicate the visibility of the image. The ServerChange event can be used to catch the change in value of the control on the server.

Properties		
Attributes	Checked	ClientID
Controls	Disabled	EnableViewState
ID	Name	NamingContainer
Page	Parent	Site
Style	TagName	TemplateSourceDirectory
Type	UniqueID	Value
Visible		
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerChange
Unload		

HtmlInputFile

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

Provides a way for you to upload files to the server. This control encapsulates the <input type=file> tag on the client and also provides a way to extract the file information from the posted data. For this control to work, the EncType of the form must be set to multipart/form-data.

Properties		
Accept	Attributes	ClientID
Controls	Disabled	EnableViewState
ID	MaxLength	Name
NamingContainer	Page	Parent

This document is created with the unregistered version of CHM2PDF Pilot

PostedFile	Site	Size
Style	TagName	TemplateSourceDirectory
Type	UniqueID	Value
Visible		
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

Listings 2.20 and 2.21 show a page that collects the name of a file from the user and then uploads it to the server. On the server side we grab the content of the posted file and place it into a text area. The Accept property is used to indicate that only files with a MIME type of text are allowed to be uploaded.

Listing 2.20 The HTML for inputfile.aspx

```
<%@ Page language="c#" Codebehind="InputFile.aspx.cs" AutoEventWireup="false"
Inherits="HtmlControls.InputFile" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Input File Example</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="InputFile" method="post" runat="server" enctype="multipart/
form-data">
            <input type="file" id="FilePost" runat="server" NAME="FilePost"><input
type="submit" id="btnSubmit" runat="server" value="Send File" NAME="btnSubmit">
            <br>
            <textarea id="txtOutput" runat="server" style="WIDTH: 733px; HEIGHT:
630px" rows="39" cols="89" NAME="txtOutput"></textarea>
        </form>
    </body>
</HTML>
```

Listing 2.21 The Code for inputfile.aspx

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
namespace HtmlControls
{
    /// <summary>
    /// Summary description for InputFile.
    /// </summary>
    public class InputFile : System.Web.UI.Page
    {
        protected System.Web.UI.HtmlControls.HtmlInputFile FilePost;
        protected System.Web.UI.HtmlControls.HtmlInputButton btnSubmit;
        protected System.Web.UI.HtmlControls.HtmlTextArea txtOutput;

        private void Page_Load(object sender, System.EventArgs e)
        {
            FilePost.Accept = "text/*";
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.btnSubmit.ServerClick += new
System.EventHandler(this.btnSubmit_ServerClick);
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void btnSubmit_ServerClick(object sender, System.EventArgs e)
        {
            System.IO.StreamReader tr = new System.IO.StreamReader(FilePost.
PostedFile.InputStream);

            txtOutput.Value = tr.ReadToEnd();

        }
    }
}
```

HtmlInputHidden

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlInputHidden class encapsulates the <input type=hidden> tag. This can be used to put hidden text into the body of a form.

This document is created with the unregistered version of CHM2PDF Pilot

NOTE

In Web programming, it was common to use a <hidden> control to retain state information from one page reload to the next. It's not as common to use the HtmlInputHidden control for this purpose in ASP.NET because you have so many other options for state management. For example, you might want to use the State bag provided by the ViewState property of the Page object instead.

Properties		
Attributes	ClientID	Controls
Disabled	EnableViewState	ID
Name	NamingContainer	Page
Parent	Site	Style
TagName	TemplateSourceDirectory	Type
UniqueID	Value	Visible
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerChange
Unload		

Listing 2.22 shows an example of using an HtmlInputHidden object to submit additional information along with the form. In this example, you want the form to send the date and time the user accessed the page along with the data the user enters. The hidden control stores and submits this additional information.

Listing 2.22 Using an HtmlInputHidden Control to Submit Additional Information in a Form

```
<%@ Page language='C#' debug='true' trace='false' %>
<script runat='server'>
  void Page_Load(Object Sender, EventArgs e)
  {
    if(!Page.IsPostBack)
      CreationDate.Value = DateTime.Now.ToString();
  }

  void btnSubmit_Click(Object Sender, EventArgs e)
  {
    spnResult.InnerHtml = "This account was created on " + CreationDate.Value;
  }
</script>
<html>
  <head>
    <title>ASP.NET Page</title>
  </head>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  <body bgcolor="#FFFFFF" text="#000000">
    <form runat='server'>
      Your Name:
      <input type="text" id="txtValue" value="Jeffrey" runat='server'
NAME="txtValue">
      <input type='submit' id="btnSubmit" OnServerClick='btnSubmit_Click'
             value="Create" runat='server' NAME="btnSubmit">
      <br>
      Your Address: <input type="text" name="txtAddress" value="4905 Brown Valley
Lane">
      <input type="hidden" id="CreationDate" runat='server' NAME="CreationDate">
    </form>
    <span id='spnResult' runat='server'></span>
  </body>
</html>
```

In this code, the current date and time is stored in the hidden control when the page is loaded. When the form is submitted to the server, the value of the date stored in the hidden field is sent to the server where it can be utilized in code. In this case, the date is simply displayed, but you could incorporate it into database insertions and so forth.

There are several alternate ways you can hide information on the page the way a hidden control does. For example, most server controls have a Visible property. Setting this property to false hides the control, enabling you to assign data to the control without the data displaying on the page.

Note that "hiding" information by assigning it to an HtmlInputHidden control doesn't actually prevent the user from accessing the information; if you view the page's source in the browser, it's easy to see the value of the hidden control. It's even conceivable that a user could change the value of the hidden control. For this reason, be careful when using hidden controls to store certain types of sensitive information in your Web applications.

For other non-visible controls, the information contained within them is not sent to the browser. Instead it is stored in ViewState. HtmlInputHidden is the one exception to this.

HtmlInputImage

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlInputImage class encapsulates the <input type=image> tag. Use this tag when you want your button to look like something other than a button. You supply the image. The ServerClick method fires an action on the server.

Properties		
Align	Alt	Attributes
Border	CausesValidation	ClientID
Controls	Disabled	EnableViewState

This document is created with the unregistered version of CHM2PDF Pilot

ID	Name	NamingContainer
Page	Parent	Site
Src	Style	TagName
TemplateSourceDirectory	Type	UniqueID
Value	Visible	
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerClick
Unload		

HtmlInputRadioButton

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlInputRadioButton class encapsulates the <input type=radio> tag. You group radio buttons together to form a group by assigning the same Name property to each button. The user may select only one member of the group.

Properties		
Attributes	Checked	ClientID
Controls	Disabled	EnableViewState
ID	Name	NamingContainer
Page	Parent	Site
Style	TagName	TemplateSourceDirectory
Type	UniqueID	Value
Visible		
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

This document is created with the unregistered version of CHM2PDF Pilot

HtmlInputText

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlInputText class encapsulates the <input type=text> tag. See Listings 2.16 and 2.17 for an example of this class in use.

Properties		
Attributes	ClientID	Controls
Disabled	EnableViewState	ID
MaxLength	Name	NamingContainer
Page	Parent	Site
Size	Style	TagName
TemplateSourceDirectory	Type	UniqueID
Value	Visible	
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerChange
Unload		

HtmlSelect

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlSelect class is the ASP.NET HTML control abstraction of the HTML SELECT element. You can set the Multiple property to true to enable the user to select multiple items in the list. The Items collection contains the items. Use <controlname>.Items[<controlname>.SelectedIndex].Value to get the selected items value. See Listing 2.9 for the HtmlSelect class in action.

Properties

This document is created with the unregistered version of CHM2PDF Pilot

Attributes	ClientID	Controls
DataMember	DataSource	DataTextField
DataValueField	Disabled	EnableViewState
ID	InnerHtml	InnerText
Items	Multiple	Name
NamingContainer	Page	Parent
SelectedIndex	Site	Size
Style	TagName	TemplateSourceDirectory
UniqueID	Value	Visible
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerChange
Unload		

HtmlTable

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlTable class encapsulates the <table> tag. The Rows property returns a collection of all the <TR> tags in the table.

Properties		
Align	Attributes	BgColor
Border	BorderColor	CellPadding
CellSpacing	ClientID	Controls
Disabled	EnableViewState	Height
ID	InnerHtml	InnerText
NamingContainer	Page	Parent
Rows	Site	Style
TagName	TemplateSourceDirectory	UniqueID
Visible	Width	
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType

This document is created with the unregistered version of CHM2PDF Pilot

HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

HtmlTableCell

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlTableCell class encapsulates the individual cells in an HtmlTableRow. The ColSpan and RowSpan properties can be used to span a number of columns or rows, respectively. The NoWrap property can be used to indicate that a cell shouldn't wrap. The Align and VAlign properties can be used to control alignment.

Properties		
Align	Attributes	BgColor
BorderColor	ClientID	ColSpan
Controls	Disabled	EnableViewState
Height	ID	InnerHtml
InnerText	NamingContainer	NoWrap
Page	Parent	RowSpan
Site	Style	TagName
TemplateSourceDirectory	UniqueID	VAlign
Visible	Width	
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

HtmlTableCellCollection

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

This document is created with the unregistered version of CHM2PDF Pilot

This class represents a collection of all HtmlTableCells within an HtmlTable control.

Properties		
Count	IsReadOnly	IsSynchronized
Item	SyncRoot	
Methods		
Add	Clear	CopyTo
Equals	GetEnumerator	GetHashCode
GetType	Insert	Remove
RemoveAt	ToString	

HtmlTableRow

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlTableRow class encapsulates a <tr> tag.

Properties		
Align	Attributes	BgColor
BorderColor	Cells	ClientID
Controls	Disabled	EnableViewSTate
Height	ID	InnerHtml
InnerText	NamingContainer	Page
Parent	Site	Style
TagName	TemplateSourceDirectory	UniqueID
VAlign	Visible	
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	Unload

HtmlTableRowCollection

This document is created with the unregistered version of CHM2PDF Pilot

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

A collection of the table rows inside the HtmlTable control.

Properties		
Count	IsReadOnly	IsSynchronized
Item	SyncRoot	
Methods		
Add	Clear	CopyTo
Equals	GetEnumerator	GetHashCode
GetType	Insert	Remove
RemoveAt	ToString	

HtmlTextArea

Member of System.Web.UI.HtmlControls.

Assembly: System.Web.dll.

The HtmlTextArea class encapsulates the <textarea> tag. The Rows and Cols properties can be used to dynamically size the control. See Listing 2.20 for an example of the HtmlTextArea control in action.

Properties		
Attributes	ClientID	Cols
Controls	Disabled	EnableViewState
ID	InnerHtml	InnerText
Name	NamingContainer	Page
Parent	Rows	Site
Style	TagName	TemplateSourceDirectory
UniqueID	Value	Visible
Methods		
DataBind	Dispose	Equals
FindControl	GetHashCode	GetType
HasControls	RenderControl	ResolveUrl
ToString		
Events		
DataBinding	Disposed	Init
Load	PreRender	ServerChange

This document is created with the unregistered version of CHM2PDF Pilot

Unload		

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Attributes of the Page Object

In ASP.old, you wrote the majority of your page-rendering code against five objects (Application, Request, Response, Server, and Session). All these objects exist in ASP.NET, although their relative utility is diminished somewhat by the more structured event-driven paradigm provided by ASP.NET.

For example, in ASP.old you typically built applications based on forms that were submitted to the server and handled in script. The script automatically parsed the elements of the form into members of the Request.Form collection. You could then send data back to the browser as a combination of templated HTML and calls to the Write method of the Response object.

In ASP.NET, you don't have to use Request.Form to read the contents of form controls; instead, you can read the properties of those controls directly. This makes programming much easier, eliminating a conceptual hurdle to building sophisticated user interfaces and ensuring that data handled by your application is strongly typed from end to end. You do not need to use Response.Write to send output to the browser, either. Although you may be accustomed to using this quite frequently in ASP.old, you'll almost never see it in ASP.NET.

The following sections describe elements of the Page object in more depth, including the familiar Request, Response, Server, and Session objects, and adding some new functionality provided by the Page object that's unique to ASP.NET.

Page Directives

Page directives are commands, inserted at the top of an ASP.NET page, that represent a mixed bag of settings pertaining to how the page is rendered and processed.

Table 2.1 shows a complete list of ASP.NET page directives.

Table 2.1. ASP.NET Page Directives

Directive	Description
@Page	A mixed bag of settings pertaining to how the page is rendered, buffered, globalized, and so forth.
@Control	Settings specific to how user controls are rendered. This setting is appropriate for user controls only.
@Import	Imports a namespace.
@Implements	Utilizes an externally defined user control, server control, or COM interface.
@Register	Registers a server control tag prefix and namespace for use in the page.
@Assembly	Links an assembly to the page.

This document is created with the unregistered version of CHM2PDF Pilot

@OutputCache	Determines how the page caches output.
@Reference	Links a page or user control to the current page.

You insert a page directive at the top of the page. Listing 2.23 shows an example of a typical @Page directive.

Listing 2.23 Typical @Page Directive

```
<%@ Page language="C#" debug="true" trace="true" %>
```

This Page directive instructs ASP.NET to interpret code in the page as C#, to activate debugging, and to execute in Trace mode to assist with debugging and performance analysis.

NOTE

The @Page directive is used when you want to change the default settings for a single page in your ASP.NET Web application. However, some of the settings in the @Page directive can also be altered for an entire directory (using the Web.config settings file) or an entire server (using the Machine.config settings file). See Chapter 5, "Configuration and Deployment," for more information on how to use these files to configure your server.

The next few sections give examples and scenarios that demonstrate and describe when you would use the various page directives. (The debugging and tracing features of ASP.NET are covered in Chapter 3, "Debugging ASP.NET Applications.")

Setting Single-Thread Compatibility Using the AspCompat Attribute

COM components built with Visual Basic 6.0 use single-threaded apartment (STA) threading. To use an STA component with ASP.NET, your ASP.NET page must also use single-threaded apartments.

Setting the AspCompat attribute of the @Page directive to true causes your page to execute on a single thread, ensuring compatibility with STA components. It also makes objects contained by the Page such as Request and Response available to any COM objects utilized in script on the page.

This attribute should be used sparingly because it degrades ASP.NET performance. Use it only in cases where you are unable to port the STA object to .NET.

Controlling Event Handlers Using the AutoEventWireup Attribute

The AutoEventWireup attribute of the @Page directive is used to override the default event procedures used to handle Page events. This attribute is set to true by default.

In general, most of the time this will have any bearing only on the name of the procedure used to handle the Page object's Load event. When AutoEventWireup is true, the event procedure is called Page_Load. If AutoEventWireup

This document is created with the unregistered version of CHM2PDF Pilot

is false, you have to create a custom event handler to handle the Page object's events.

This feature is used most often with code behind. In order to make the events fire when AutoEventWireup is set to false, define delegates for Page_Init in the constructor of the code-behind class. Define delegates for all events in the Page_Init(). Listing 2.24 shows an example of the scenario.

Listing 2.24 Definition of a Code-Behind Page_Load Event Procedure Without AutoEventWireup

```
public MyPage()
{
  this.Init += new System.EventHandler(this.Page_Init);
}
private void Page_Init(object Sender, EventArgs e)
{
  this.Load += new System.EventHandler(this.MyPageLoad);
}
private void MyPageLoad(object Sender, EventArgs e)
{
  //Todo: enter your code here
}
```

You can see that the Page_Init() is explicitly wired to the event Page_Init by the delegate in the constructor. Then the Page_Init() event handler is used to define delegates for other events. This also allows us to associate a subroutine with any name with a specified event.

For more information on how code behind works, see the section "Separating Presentation from Code Using Code Behind" earlier in this chapter.

Deactivating Page Buffering Using the Buffer Attribute

When the Web server is building a page for display in the browser, it can either send the data to the browser a little at a time, or it can store the entire page and send it to the browser in one fell swoop. Sending the data to the browser all at once is called buffering. Buffering a page can yield significant performance improvements because it potentially lowers the number of TCP packets that must be sent to return the page to the user. It can also make for a more positive perceived user experience because a buffered page renders all at once instead of progressively painting as data arrives.

In ASP.NET, buffering is turned on by default. Setting the Buffer attribute in the @Page directive to false turns off buffering.

Denoting the Page's Class Name Using the ClassName Attribute

You use the ClassName attribute to denote the name of the class used by the page when it is compiled. This attribute is commonly set by Visual Studio .NET in conjunction with the code-behind feature.

For more information on how code-behind works, see the section "Separating Presentation from Code Using Code Behind" earlier in this chapter.

This document is created with the unregistered version of CHM2PDF Pilot

Specifying the Target Client Using the ClientTarget Attribute

You can use the ClientTarget attribute to specify the user agent (a.k.a. browser type) for which the server controls in your application should render their content. It makes sense to use this option in a situation where you have a captive audience that has standardized on a particular browser (as in an intranet), and you are using server controls that adapt to browser capabilities.

NOTE

The ASP.NET documentation doesn't explicitly state that setting this value produces a performance gain, but presumably it would because server controls would not have to sniff the browser before rendering themselves.

Every browser is supposed to pass a user agent string identifying the type and version number of the browser each time it makes an HTTP request. You can programmatically retrieve the user agent string reported by the browser by inspecting the value of Page.Request.UserAgent.

Note that some browsers try to trick the Web server into thinking that they're a different kind of browser by passing a bogus user-agent string. For example, by default, Opera 5.0 identifies itself to the browser as Internet Explorer 5.0. This is done to ensure that the browser will work with Web applications that sniff the browser in brain-dead ways (for example, by attempting to detect the brand of the browser instead of its underlying capabilities).

Setting the Language Using the CodePage Attribute

A code page is a set of mappings between text characters and numbers.

One common code page in the United States is UTF-8, also known as Unicode, and described in Internet RFC 2279 (see http://www.ietf.org/rfc/rfc2279.txt if you're really interested).

It's necessary only to specify the CodePage attribute when the page you've authored was written using a different code page than the Web server it's running on. We're hoping you don't do this too often.

Setting Compiler Options Using the CompilerOptions Attribute

You can use the CompilerOptions attribute to pass additional arguments to the compiler when the page is run. To do this, you'll need to have an excellent handle on how compiler switches work, a page that somehow lacks a capability found in a compiler switch, and fortitude.

We racked our brains trying to figure out why you would use this feature, but we couldn't come up with much. The documentation isn't much help, either. Many of the compiler switches are already represented in ASP.NET in various ways, so it's possible that the CompilerOptions is simply a hook that ASP.NET provides for you, enabling you to

This document is created with the unregistered version of CHM2PDF Pilot

take advantage of future compiler options that may become available before ASP.NET officially supports them.

Setting the ContentType Attribute

The ContentType attribute maps to an HTTP setting that tells the browser what kind of data to expect as the response to a request. Almost always, the data sent from the Web server will be an HTML page. However, you may want to change this. You can use the ContentType attribute of the Page directive to make this change.

You change the ContentType attribute in situations where your ASP.NET page is designed to send data other than HTML to the browser; one common, real-world situation would be sending XML directly to the browser. To do this, set the content type to "text/xml". If the output of the page contains well-formed XML data (and nothing but XML data), and your browser has the capability to display XML directly (that is, Internet Explorer 5.0 or later), the XML will render directly in the browser.

Specifying International Culture Using the Culture Attribute

You can use the Culture attribute to specify for which culture the content of your Web page is intended. Culture in this context means international dialect and language. For example, the culture attribute "en-US" stands for U.S. English, whereas "en-GB" stands for the kind of English that our good friends in Great Britain speak.

Certain operations in the .NET framework, such as the formatting of strings, are culture dependent. For example, many European cultures use a comma instead of the decimal point used by Americans and other sensible cultures. (Just kidding.)

Activating Debugging Using the Debug Attribute

Setting the Debug attribute to true activates ASP.NET Debug mode, which provides rich debugging information in the browser to remote machines when an error is encountered in an ASP.NET page.

Debugging is covered in Chapter 3.

Setting the Description Attribute

The Description attribute enables you to append a textual description of your choice to your page. This attribute isn't accessible programmatically; it's presumably just a way for you to insert a comment describing the page in the @Page directive.

Setting the EnableSessionState Attribute

Session state refers to the capability of a Web application to store information for individual users as the user navigates from page to page. Session state is turned on by default; you may want to consider setting the EnableSessionState attribute to false—that deactivates Session state and can increase performance.

This document is created with the unregistered version of CHM2PDF Pilot

See Chapter 4, "State Management and Caching," for a general discussion of how Session state works in ASP.NET.

Activating View State Using the EnableViewState and EnableViewStateMac Attribute

View state is the feature of ASP.NET that causes a control's properties to be retained across round trips to the server. It is discussed more fully in the section "Taking Advantage of Postback and View State," later in this chapter.

View state is enabled in ASP.NET pages by default. Setting the EnableViewState attribute to false enables you to turn View state off. Note that it is also possible to turn View state off on a control-by-control basis, for controls that support it, by setting the control's ViewStateEnabled property to false.

Turning off View state can increase application performance by reducing the amount of data that must be sent to and from the server.

Setting EnableViewStateMac to true enables an additional check to ensure that View state information was not altered between the time it was sent to the browser by the server and the time it was resubmitted to the server (Mac in this context stands for Machine Authentication Check). This is an important security feature that you should employ whenever sensitive data is stored in View state.

Setting the ErrorPage Attribute

The ErrorPage attribute enables you to redirect to a custom error page of your choosing. The value for the attribute can be any URL. This attribute is commonly set in applications where it's likely that the user will enter a bogus value (a long URL, for example, or perhaps a mistyped query string parameter), and you don't want the user to be confronted with a grungy ASP.NET error page.

Inheriting from a Class Using the Inherits Attribute

Each ASP.NET page is treated as a class. You can cause your ASP.NET page to inherit from another class by setting the Inherits attribute.

You typically use the Inherits attribute to take advantage of code-behind functionality. Code-behind functionality is described in more detail in the section "Separating Presentation from Code Using Code Behind" in this chapter.

Setting the Language Attribute

The Language attribute determines the programming language used in the page. By default, you can choose VB, C#, or JScript, although other .NET languages could be used as well. Note that an ASP.NET page can only have one language.

Setting the Locale Identifier Using the LCID Attribute

The locale identifier (LCID) is an integer that corresponds to a national language setting. The idea behind an LCID is

This document is created with the unregistered version of CHM2PDF Pilot

to give the client some idea of what national language the application supports. Because the LCID value is a 32-bit number, the value can be quite granular when describing different dialects of the same language. For example, the LCID value 1033 denotes United States English; the value 2057 denotes the flavour of English spoken in the United Kingdom.

By default, ASP.NET uses your Web server's locale (set in operating system configuration) as the locale for each page. Setting this attribute to a different value overrides the locale setting.

Setting the Src Attribute for Code Behind

The code executed by a particular ASP.NET source file can be located in an external file. You use the Src (source) attribute to specify the name of the external file.

You typically use the Src attribute to take advantage of code-behind functionality. Code-behind functionality is described in more detail in the section "Separating Presentation from Code Using Code Behind" in this chapter.

Enabling Internet Explorer Smart Navigation Features

Setting the SmartNavigation attribute to true enables certain usability features specific to Microsoft Internet Explorer 5.0 and later. In particular, Smart Navigation can improve the user experience by preserving the scroll position on the page and is therefore appropriate in situations where your page contains a long form that is posted back repeatedly.

Setting the Trace Attribute to Activate Tracing

Setting the Trace attribute to true activates tracing for the current page. Tracing helps developers debug an ASP.NET application. Your trace code might simply indicate that a particular piece of code executed successfully; trace code can also give you a sense of how control flows as a page executes and how long each operation in the lifetime of a page takes to accomplish.

Tracing is covered in more detail in Chapter 3.

Setting the TraceMode Attribute to Sort Trace Information

The TraceMode attribute determines how trace information is displayed when tracing is turned on. Setting TraceMode to SortByTime displays trace entries from earliest to latest. Setting the attribute to SortByCategory groups trace entries by type.

Tracing is covered in more detail in Chapter 3.

Setting the Transaction Attribute to Support Transactions

You can add support for transactions to your pages by using the Transaction attribute. Transactions are set to Disabled by default; other settings are NotSupported, Supported, Required, and RequiresNew.

This document is created with the unregistered version of CHM2PDF Pilot

Denoting the Culture with the UICulture Attribute

You can give your ASP.NET page the ability to display culture-specific actions by setting the UICulture attribute. The UICulture attribute differs from the Culture attribute (described earlier in this section) in the sense that while Culture has a stand-alone effect on the display of data (such as date formats and decimalization) in the page, UICulture is simply a marker that enables you to denote a culture or language for later use in your localization code. How you provide the culture-appropriate string resource based on the value of UICulture is up to you.

Setting the WarningLevel Attribute

You can force the .NET compiler to treat compiler warnings as errors by setting the WarningLevel attribute of the @Page directive to true.

Five levels of compiler warnings exist, numbered 0 through 4. When the compiler transcends the warning level set by this attribute, compilation fails. The meaning of each warning level is determined by the programming language and compiler you're using; consult the reference specification for your compiler to get more information about the warning levels associated with compiler operations and what triggers compiler warnings.

@Control Directives

You use an @Control directive in place of an @Page directive in a user control (.ASCX) file. User controls are script files that provide programmable user-interface functionality. The @Control directive has many of the same attributes as the @Page directive.

User controls are discussed in Chapter 9, "Building User Controls and Server Controls."

@Import Directives

The @Import directive is used to make classes found in that namespace easier to access. When you import a namespace, you don't have to refer to the full namespace syntax to refer to a class in that namespace; you can, for example, use the name DataSet instead of System.Data.Dataset.

An Import directive looks like this:

```
<@ Import namespace="System.Data" %>
```

You can have as many Import directives in your page as you want; each namespace reference should have its own Import directive.

Note that you can use classes in any namespace loaded into the Global Assembly Cache (GAC); this includes all .NET framework classes and anything else you've stuck in the GAC.

This document is created with the unregistered version of CHM2PDF Pilot

Implementing External Interfaces with the @Implements Directive

You use the @Implements directive to implement an interface. To understand why this feature exists, remember that every ASP.NET page is really a kind of subclass of the Page object. If you want the object to have access to functionality that requires the implementation of an interface, you must use this directive.

For example, you could use @Implements in user-control development to implement the IPostBackEventHandler interface as a way of raising events associated with postback.

You can't use the @Implements directive in a code-behind class (specify interfaces by using : in the class declaration).

Registering Controls with the @Register Directive

The @Register directive is used to make your ASP.NET page aware of user controls and server controls. It also gives custom ASP.NET controls a programmer-defined namespace, ensuring that the name of one control doesn't conflict with the name of another control.

Custom ASP.NET user interface controls are discussed in Chapter 9.

@Assembly Directives

The @Assembly directive is used to make your ASP.NET page aware of external components. You can use one of two elements with this directive—Name or Src. Name is the name of a precompiled assembly (without the .DLL extension). Src represents the name of a source code file (such as myclass.cs).

@OutputCache Directives

You can use @OutputCache directives to control how page caching works. Page caching is a feature that can improve performance on ASP.NET pages whose content changes infrequently.

Caching is discussed in more detail in Chapter 4.

Events Raised by the Page Object

The Page object raises a set of events that you can respond to in code.

The events raised by the Page object are standard .NET events, which means they receive a minimum of two arguments: a Sender (a reference to the object that raised the event) and a collection that represents arguments passed to that event (inherited from System.EventArgs). The contents of this collection differ, depending on the event procedure.

This document is created with the unregistered version of CHM2PDF Pilot

Table 2.2 shows a list of events supported by the Page object.

Table 2.2. Events Supported by the Page Object

Event	**Description**
AbortTransaction	A transaction was aborted.
CommitTransaction	A transaction was completed.
DataBinding	Data binding was completed.
Disposed	The page has been released from memory. This is the last event.
Error	An unhandled exception was thrown on the page.
Init	The page was first loaded. This is the first event raised by the page.
Load	The page is loaded.
PreRender	The page content is about to be displayed.
Unload	The page is about to be unloaded from memory.

As you've seen in the code examples to this point, the most common event handled in the Page object is Load.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Creating User Interfaces with Web Controls

The .NET framework provides a number of user-interface components that you can use.

These components are divided into several categories:

-
- HTML controls—An object-oriented way of creating standard HTML page elements such as text boxes and buttons.
-
-
- Web forms controls—Objects that can incorporate standard and dynamic HTML into rich user-interface elements.
-
-
- Validation controls—A type of Web form control specifically geared toward validating user input on a form, often without causing a round trip to the server. Validation controls are covered in Chapter 11, "Creating Database Applications with ADO.NET."
-

This section will show you how to use the controls that are included with the .NET framework. You can also create your own Web forms controls; this is covered in Chapter 9.

Programming Web Forms Controls

The first step to using Web forms controls on a page is to create an ASP.NET Web form. You do this by creating a conventional HTML FORM tag with the ASP.NET runat="server" attribute, as shown in Listing 2.25.

Listing 2.25 Basic ASP.NET Web Form Declaration

```
<form id='myform' method='post' runat='server'>

</form>
```

Control Event Model

Web controls raise events of their own as well as the events of the classes they inherit.

Web controls that inherit from System.Web.UI.WebControl.WebControl raise the events listed in Table 2.3.

Table 2.3. Events Raised by Web Controls

This document is created with the unregistered version of CHM2PDF Pilot

Event	Description
DataBinding	Occurs when the control binds to a data source. (For more information on data binding, see Chapter 11.)
Disposed	Occurs when the control is destroyed; this is always the last event raised by the control.
Init	Occurs when the control is first created; this is always the first event raised by the control.
Load	Occurs when the control is loaded on the page.
PreRender	Occurs when the control is about to be rendered (displayed on the screen).
Unload	Occurs when the control is unloaded from memory.

Taking Advantage of Postback and View State

Postback is the ASP.NET feature that enables you to detect whether a form has been submitted to the server. View state is a related concept that causes the contents of a posted-back form to be preserved.

The concepts of postback and View state grew out of a frustration with traditional Web server scripting paradigms. In a Web form, as with most HTTP Web requests, the page is completely destroyed and re-created each time it is submitted to the server. But in many, if not most cases, you want certain values (such as the contents of a Search box or the selection you've made in a list of custom preferences) to remain on the page. Without View state, you'd have to write code to make this happen, and without postback detection, it's tricky to determine whether the existing form data needs to be displayed.

You can determine whether an ASP.NET Web form is being posted back to the server by inspecting the IsPostBack property of the Page object. The first time a user accesses the page, IsPostBack will be false. At this time, you typically perform initialization work in code. This can be as straightforward as setting defaults in controls or as involved as performing a relational database query to serve as the basis for display on the page.

NOTE

It is possible to manipulate postback directly in code. To do this, you implement the IPostBackEventHandler interface in your page.

Earlier in this chapter, Listing 2.5 demonstrated the HTML source of an ASP.NET Web page that contains chunks of encoded View state. Listing 2.26 shows a snippet of that voluminous code listing.

Listing 2.26 HTML Source for an ASP.NET Page Highlighting View State Information

```
    <form name="WebForm1" method="post" action="calendar.aspx" id="WebForm1">
<input type="hidden" name="__VIEWSTATE" value="dDw1MzYzNjkxODU7Oz4=" />

        <table id="Calendar1" cellspacing="0" cellpadding="2" border="0"
        style="border-width:1px;border-style:solid;border-collapse:collapse;">
```

This document is created with the unregistered version of CHM2PDF Pilot

```
<tr>
  <td colspan="7" style="background-color:Silver;">
     <table cellspacing="0" border="0"
       style="width:100%;border-collapse:collapse;">
       <tr>
         <td style="width:15%;">
           <a href="javascript:__doPostBack('Calendar1','prevMonth')"
             style="color:Black">&lt;</a>
```

Note that we've mercifully deleted the majority of the source code for brevity. You won't have to write this code yourself; fortunately, ASP.NET generates it for you from the high-level ASP.NET controls you specify.

You can see that the View state information is embedded in the page as a hidden control with the name __VIEWSTATE. The value is encoded, but not encrypted.

You should also be able to see that a hyperlink (a tag) toward the end of the code snippet is attached to an automatically generated JavaScript procedure called __doPostBack. This procedure parses the contents of View state and submits the form back to the server.

The ultimately cool thing about View state is that you don't have to lift a finger to make it work. You may choose to turn it off, however; getting rid of View state when you don't need it can provide a bit of a performance gain. To turn View state off for a particular control, set the control's ViewState property to false. (View state can also be shut off for a particular page or an entire Web application by using @Page dir-ectives, described earlier in this chapter, and Web.config settings, described in Chapter 5, "Configuration and Deployment.")

Mobile Controls

The Microsoft.NET framework supports devices with small form factors and limited capabilities, such as mobile phones and Pocket PCs. These devices have demands on the user-interface developer that are totally different from the challenges presented by user-interface design on a desktop PC.

An in-depth discussion of mobile Web forms is beyond the scope of this book; however, many of the techniques discussed in this chapter will give you a good foundation for creating applications for mobile devices. Because the technical implementation (involving HTML, Dynamic HTML, WAP, and so forth) is abstracted behind the .NET Base Class Library, you can use similar programming techniques to code for Windows and Windows CE.

Data Binding

It is possible to bind a data source to controls in ASP.NET. This works both for rich Web forms controls and HTML controls.

For more information on data binding, see Chapter 11.

Determining Browser Capabilities

This document is created with the unregistered version of CHM2PDF Pilot

When you're creating a corporate intranet, you can often specify corporate standards for a Web browser. Although this tends to limit user choice, this is useful for the developer because it means you can utilize the capabilities of a specific Web browser.

For Web sites that are intended to be used by the general public, though, you can't know ahead of time what kind of browser the users will have. In this case, you may find it useful to have a programmatic way of determining things about the user's browser. For example, does the user's browser support JavaScript (or, perhaps, did the user deactivate JavaScript support)? How about cookies? Even though most modern browsers support them, users can choose to turn those off, too.

The HttpBrowserCapabilities class, found in the System.Web namespace, provides these capabilities in ASP.NET. An instance of an HttpBrowserCapabilities object is contained by the ASP.NET Request object (in this context an instance of the object, called Browser, is created for you automatically when a page is requested from the server). Listing 2.27 shows how to use this pre-created object to display information about the browser.

Listing 2.27 Complete Listing of All HttpBrowserCapabilities Contained in the Request Object

```
<%@ PAGE language='c#' debug='true' trace='false' %>

<HTML>
<HEAD>
<TITLE>ASP.NET Browser Capabilities</TITLE>
</HEAD>

<BODY>
<table width="350" border="0" cellspacing="1" cellpadding="3"
   bgcolor="#000000">
  <tr bgcolor="#FFFFFF">
    <td>Supports ActiveX Controls</td>
    <td><% =Request.Browser.ActiveXControls %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Is an America Online Browser</td>
    <td><% =Request.Browser.AOL %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Supports background sounds</td>
    <td><% =Request.Browser.BackgroundSounds %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Is a beta version browser</td>
    <td><% =Request.Browser.Beta %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Browser name (user-agent)</td>
    <td><% =Request.Browser.Browser %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Supports Channel Definition Format</td>
    <td><% =Request.Browser.CDF %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Common Language Runtime version</td>
    <td><% =Request.Browser.ClrVersion %></td>
  </tr>
  <tr bgcolor="#FFFFFF">
    <td>Cookies available</td>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  <td><% =Request.Browser.Cookies %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Is this a Web search engine (&amp;quot;crawler&amp;quot;)?</td>
  <td><% =Request.Browser.Crawler %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Version of JavaScript (ECMAScript) supported</td>
  <td><% =Request.Browser.EcmaScriptVersion %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Supports frames</td>
  <td><% =Request.Browser.Frames %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Supports client-side Java</td>
  <td><% =Request.Browser.JavaApplets %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Supports JavaScript (ECMAScript)</td>
  <td><% =Request.Browser.JavaScript %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Browser version</td>
  <td>
    <% =Request.Browser.MajorVersion + "." +
     Request.Browser.MinorVersion %>
  </td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Microsoft XML Document Object Model version</td>
  <td><%=Request.Browser.MSDomVersion %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Operating system platform</td>
  <td><% =Request.Browser.Platform %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Supports HTML tables</td>
  <td><% =Request.Browser.Tables %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Client browser type</td>
  <td><% =Request.Browser.Type %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Browser supports VBScript</td>
  <td><% =Request.Browser.VBScript %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Version of client browser</td>
  <td><% =Request.Browser.Version %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>W3C HTML Document Object Model version</td>
  <td><% =Request.Browser.W3CDomVersion %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Running 16-bit Windows?</td>
  <td><% =Request.Browser.Win16 %></td>
</tr>
<tr bgcolor="#FFFFFF">
  <td>Running 32-bit Windows?</td>
  <td><% =Request.Browser.Win32 %></td>
</tr>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
</table>
</BODY>
</HTML>
```

I used godless, archaic render blocks to create this page, mainly to make it easier for me to create, but also to make it simpler to read.

If you have more than one kind of browser installed on your computer, you may find it interesting to navigate to this page using both of them to see the different capabilities reported by the Browser object. For example, I found it interesting to learn that JavaScript 1.2 comes with Internet Explorer 6.0 beta, but that Opera 5.0 comes with the (presumably bigger and badder) JavaScript 1.3. Also, Opera doesn't return operating system platform information to the server the way Internet Explorer does. It's just as well; with well-designed Web applications, the server shouldn't need to know what operating system the browser is running on anyway.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Server Controls and Page Object Reference

This section provides a quick reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book. For the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

Validation controls are covered in Chapter 11.

This chapter covers the Page framework classes in ASP.NET, including the Page object itself, its children, and user-interface controls (HTML controls and server controls). The following sections provide a brief reference to the properties, methods, and events provided by those classes.

AdRotator Class

Member of System.Web.UI.WebControls.

The AdRotator class enables you to display a graphical advertisement on your Web page, changing (or "rotating") the advertisement from a list of graphical URLs. Because they are stored in the form of URLs, the graphics and the pages they link to can reside anywhere on the Web.

Properties		
AccessKey	CssClass	Site
AdvertisementFile	Enabled	Style
BackColor	EnableViewState	TabIndex
BorderColor	Font	Target
BorderStyle	ForeColor	TemplateSourceDirectory
BorderWidth	Height	ToolTip
BackColor	ID	UniqueID
ClientID	KeywordFilter	Visible
Controls	NamingContainer	Width
ControlStyle	Page	
ControlStyleCreated	Parent	
Methods		
AddAttributesToRender	GetHashCode	OnUnload
AddParsedSubObject	GetType	RaiseBubbleEvent
ApplyStyle	HasControls	Render
ClearChildViewState	IsLiteralContent	RenderBeginTag
CopyBaseAttributes	LoadViewState	RenderChildren

This document is created with the unregistered version of CHM2PDF Pilot

CreateChildControls	MapPathSecure	RenderContents
CreateControlCollection	MemberWiseClone	RenderControl
CreateControlStyle	MergeStyle	RenderEndTag
DataBind	OnAdCreated	ResolveUrl
Dispose	OnBubbleEvent	SaveViewState
EnsureChildControls	OnDataBinding	ToString
Equals	OnInit	TrackViewState
Finalize	OnLoad	
FindControl	OnPreRender	
Events		
AdCreated	Init	UnLoad
DataBinding	Load	
Disposed	PreRender	

The list of advertisements is stored in an XML file. Listing 2.28 shows an example of an XML file.

Listing 2.28 Using the AdRotator Class

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
    <Ad>
        <ImageUrl>ad-1.png</ImageUrl>
        <NavigateUrl>http://www.redblazer.com/</NavigateUrl>
        <AlternateText>Advertisement Number One</AlternateText>
    </Ad>
    <Ad>
        <ImageUrl>ad-2.png</ImageUrl>
        <NavigateUrl>http://www.redblazer.com/</NavigateUrl>
        <AlternateText>Advertisement Number Two</AlternateText>
    </Ad>
</Advertisements>
```

The ImageUrl node denotes which image to display. The NavigateUrl note indicates the page to which the advertisement is linked. The value of the AlternateText node is displayed if the browser has graphics turned off in the browser; it's also displayed as a pop-up tool tip for the image.

You can have as many Ad elements as you want; the AdRotator will randomly switch between them. To see this, reload the page several times. (To avoid viewing a cached version of the page, be sure you press Ctrl+F5 rather than using the View, Refresh menu command in your browser.)

NOTE

The standard dimensions for a Web advertising banner is 468 pixels wide by 60 pixels high. You can, of course, use the AdRotator control to display images of any dimensions you want.

You link an XML file with an instance of an AdRotator control by assigning the name of the file to the AdRotator

This document is created with the unregistered version of CHM2PDF Pilot

control's AdvertisementFile property. Listing 2.29 shows an example.

Listing 2.29 Using an AdRotator Control to Display Various Advertising Banners in an ASP.NET Page

```
<%@ Page Language="c#" debug="False" %>
<HTML>
  <HEAD>
    <title>ASP.NET AdRotator Control</title>
  </HEAD>
  <body>
    <form runat="server">
      <asp:AdRotator id="AdRotator1" runat="server" Width="468px" Height="60px"
AdvertisementFile="ads.xml" />
    </form>  </body>
</HTML>
```

As you can see, no event procedure code is necessary to use the AdRotator control (although the control does provide an AdCreated event you can use to intercept and reassign the properties of the advertisement before it is rendered on the page).

Button Class

Member of System.Web.UI.WebControls.

The Button class is the ASP.NET Web control equivalent of the HTML submit and button elements. The class abstracts both elements, adding a number of additional properties and events as well.

Properties		
AccessKey	ControlStyle	Page
Attributes	ControlStyleCreated	Parent
BackColor	EnableViewState	Site
BorderColor	CssClass	Style
BorderStyle	Enabled	TabIndex
BorderWidth	EnableViewState	TemplateSourceDirectory
CausesValidation	Font	Text
ClientID	ForeColor	ToolTip
CommandArgument	Height	UniqueID
CommandName	ID	Visible
Controls	NamingContainer	Width
Methods		
AddAttributesToRender	GetHashCode	OnPreRender
AddParsedSubObject	GetType	OnUnload
ApplyStyle	HasControls	RaiseBubbleEvent
ClearChildViewState	IsLiteralContent	Render
CopyBaseAttributes	LoadViewState	RenderBeginTag
CreateChildControls	MapPathSecure	RenderContents

This document is created with the unregistered version of CHM2PDF Pilot

CreateControlCollection	MemberwiseClone	RenderControl
CreateControlStyle	MergeStyle	RenderEndTag
DataBind	OnBubbleEvent	ResolveUrl
Dispose	OnClick	SaveViewState
EnsureChildControls	OnCommand	ToString
Equals	OnDataBinding	TrackViewState
Finalize	OnInit	
FindControl	OnLoad	
Events		
Click	Disposed	PreRender
Command	Init	UnLoad
DataBinding	Load	

Code examples that use the Button object are given throughout this book. Essentially, only two members of the Button object are used with any regularity: the Text property, to display text on the button face, and the Click event, which generates a form submit.

Calendar Class

Member of System.Web.UI.WebControls.

The Calendar class renders a calendar in HTML. In most cases, you'll want to respondto a user selecting a date in the calendar; do this by handling the control's SelectionChanged event.

Properties		
AccessKey	Font	ShowGridLines
Attributes	ForeColor	ShowNextPrevMonth
BackColor	HasChildViewState	ShowTitle
BorderColor	Height	Site
BorderStyle	ID	Style
BorderWidth	IsTrackingViewState	TabIndex
CellPadding	NamingContainer	TagKey
CellSpacing	NextMonthText	TagName
ChildControlsCreated	NextPrevFormat	TemplateSourceDirectory
ClientID	NextPrevStyle	TitleFormat
Context	OtherMonthDayStyle	TitleStyle
Controls	Page	TodayDayStyle
ControlStyle	Parent	TodaysDate
ControlStyleCreated	PrevMonthText	ToolTip
CssClass	SelectedDate	UniqueID
DayHeaderStyle	SelectedDates	ViewState
DayNameFormat	SelectedDayStyle	ViewStateIgnoresCase

This document is created with the unregistered version of CHM2PDF Pilot

DayStyle	SelectionMode	Visible
Enabled	SelectMonthText	VisibleDate
EnableViewState	SelectorStyle	WeekendDayStyle
Events	SelectWeekText	Width
FirstDayOfWeek	ShowDayHeader	
Methods		
AddAttributesToRender	FindControl	OnSelectionChanged
AddParsedSubObject	GetHashCode	OnUnload
ApplyStyle	GetType	OnVisibleMonthChanged
BuildProfileTree	HasControls	RaiseBubbleEvent
ClearChildViewState	IsLiteralContent	Render
CopyBaseAttributes	LoadViewState	RenderBeginTag
CreateChildControls	MapPathSecure	RenderChildren
CreateControlCollection	MemberwiseClone	RenderContents
CreateControlStyle	MergeStyle	RenderControl
DataBind	OnBubbleEvent	RenderEndTag
Dispose	OnDataBinding	ResolveUrl
EnsureChildControls	OnDayRender	SaveViewState
Equals	OnInit	
Finalize	OnLoad	ToString
	OnPreRender	TrackViewState
Events		
DataBinding	Init	SelectionChanged
DayRender	Load	Unload
Disposed	PreRender	VisibleMonthChanged

Listing 2.30 shows an example of a page that displays a Calendar control and a label. When the user selects a date in the calendar, the label indicates which date the user selected.

Listing 2.30 Responding to a User Selection in the Calendar Server Control

```
<%@ Page Language="c#" debug="false"%>
<SCRIPT runat='server'>
  void Calendar1_SelectionChanged(System.Object sender, System.EventArgs e)
  {
    Label1.Text = "The date you selected is " + Calendar1.SelectedDate;
  }
</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>ASP.NET Calendar Control</TITLE>
  </HEAD>
  <body>
    <form method="post" runat="server">
      <asp:Calendar id="Calendar1" OnSelectionChanged="Calendar1_SelectionChanged"
runat="server"></asp:Calendar>
      <BR>
      <asp:Label id="Label1" runat="server"></asp:Label>
    </form>
  </body>
</HTML>
```

This document is created with the unregistered version of CHM2PDF Pilot

CheckBox Class

Member of System.Web.UI.WebControls.

The CheckBox class is the Web control abstraction of the HTML CHECKBOX element, used to enable the user to select a true/false or yes/no condition.

Properties		
AccessKey	CssClass	Style
Attributes	Enabled	TabIndex
AutoPostBack	EnableViewState	TagKey
BackColor	Events	TagName
BorderColor	Font	TemplateSourceDirectory
BorderStyle	ForeColor	Text
BorderWidth	HasChildViewState	TextAlign
Checked	Height	ToolTip
ChildControlsCreated	ID	UniqueID
ClientID	IsTrackingViewState	ViewState
Context	NamingContainer	ViewStateIgnoresCase
Controls	Page	Visible
ControlStyle	Parent	Width
ControlStyleCreated	Site	
Methods		
AddAttributesToRender	GetHashCode	OnUnload
AddParsedSubObject	GetType	RaiseBubbleEvent
ApplyStyles	HasControls	Render
ClearChildViewState	IsLiteralContent	RenderBeginTag
CopyBaseAttributes	LoadViewState	RenderChildren
CreateChildControls	MapPathSecure	RenderContents
CreateChildCollection	MemberwiseClone	RenderControl
CreateControlStyle	MergeStyle	RenderEndTag
DataBind	OnBubbleEvent	ResolveUrl
Dispose	OnCheckedChanged	SaveViewState
Equals	OnDataBinding	ToString
EnsureChildControls	OnInit	TrackViewState
Finalize	OnLoad	
FindControl	OnPreRender	
Events		
CheckedChanged	Init	Unload
DataBinding	Load	
Disposed	PreRender	

This document is created with the unregistered version of CHM2PDF Pilot

You can determine whether the check box is checked by inspecting its Boolean Checked property. Setting its Text property assigns a text label to the check box; the TextAlign property determines where the accompanying text label is displayed. Listing 2.31 shows an example.

Listing 2.31 Reading the Checked Value of an ASP.NET CheckBox Server Control

```
<%@ Page Language="c#" debug="False" %>
<HTML>
  <HEAD>
    <title>ASP.NET CheckBox Control</title>
    <SCRIPT runat='server'>
      void CheckChanged(Object Sender, EventArgs e)
      {
        if(CheckBox1.Checked)
        CheckBox1.Text = "Thank you for checking this.";
        else
        CheckBox1.Text = "Check This!";
      }
    </SCRIPT>
  </HEAD>
  <body>
    <form runat="server">
      <asp:CheckBox id="CheckBox1" runat="server" Text="Check This!"
TextAlign="Left" /><BR>
      <asp:button OnClick="CheckChanged" text="Send" runat="server" ID="Button1" />
    </form>
  </body>
</HTML>
```

Control Class

Member of System.Web.UI.

The Control class serves as the base class for all ASP.NET server controls.

Properties		
ChildControlsCreated	HasChildViewState	Site
ClientID	ID	TemplateSourceDirectory
Context	IsTrackingViewState	UniqueID
Controls	NamingContainer	ViewState
EnableViewState	Page	ViewStateIgnoresCase
Events	Parent	Visible
Methods		
AddParsedSubObject	GetHashCode	OnUnload
BuildProfileTree	GetType	OnUnload
ClearChildViewState	HasControls	RaiseBubbleEvent
CreateChildControls	IsLiteralContent	Render
CreateControlCollection	LoadViewState	RenderChildren
DataBind	MemberwiseClone	RenderControl

This document is created with the unregistered version of CHM2PDF Pilot

Dispose	OnBubbleEvent	ResolveUrl
EnsureChildControls	OnDataBinding	SaveViewState
Equals	OnInit	
Finalize	OnLoad	ToString
FindControl	OnPreRender	TrackViewState
	OnPreRender	
Events		
DataBinding	Init	PreRender
Disposed	Load	Unload

DataGrid Class

Member of System.Web.UI.WebControls.

The DataGrid class enables you to display data in a tabular (row and column) format. Like all data-bound ASP.NET controls, the DataGrid need not be bound to a traditional relational data source; you can also bind to an array or one of the many collection objects provided in the .NET framework.

The HTML output of this control is a table, but the DataGrid class adds a number of useful additional accoutrements such as data paging and formatting.

Properties		
AccessKey	CurrentPageIndex	PageCount
AllowCustomPaging	DataKeys	PagerStyle
AllowPaging	DataMember	PageSize
AllowSorting	DataSource	Parent
AlernatingItemStyle	EditItemIndex	SelectedIndex
Attributes	EditItemStyle	SelectedItem
AutoGeneratteColumns	Enabled	SelectedItemStyle
BackColor	EnableViewState	ShowFooter
BackImageUrl	Events	ShowHeader
BorderColor	Font	Site
BorderStyle	FooterStyle	Style
BorderWidth	ForeColor	TabIndex
CellPadding	GridLines	TagKey
CellSpacing	HasChildViewState	TagName
ChildControlsCreated	HeaderStyle	TemplateSourceDirectory
	Height	
ClientID	HorizontalAlign	ToolTip
Columns	ID	UniqueID
Context	IsTrackingViewState	ViewState

This document is created with the unregistered version of CHM2PDF Pilot

Controls	Items	ViewStateIgnoresCase
ControlStyle	ItemStyle	VirtualItemCount
ControlStyleCreated	NamingContainer	Visible
CssClass	Page	Width
Methods		
AddAttributesToRender	HasControls	OnPageIndexChanged
AddParsedSubObject	IsLiteralContent	OnPreRender
ApplyStyle	LoadViewState	OnSelectedIndexChanged
ClearChildViewState	MapPathSecure	OnSortCommand
CopyBaseAttributes	MemberwiseClone	OnUnload
CreateChildControls	MergeStyle	OnUpdateCommand
CreateControlCollection	OnBubbleEvent	Render
CreateControlStyle	OnCancelCommand	RenderBeginTag
DataBind	OnDataBinding	RenderChildren
Dispose	OnDeleteCommand	RenderContents
Equals	OnEditCommand	RenderControl
EnsureChildControls	OnInit	RenderEndTag
Finalize	OnItemCommand	ResolveUrl
FindControl	OnItemCreated	SaveViewState
GetHashCode	OnItemDataBound	ToString
GetType	OnLoad	TrackViewState
Events		
CancelCommand	ItemCommand	SelectedIndexChanged
DataBinding	ItemCreated	SortCommand
DeleteCommand	ItemDataBound	Unload
Disposed	Load	UpdateCommand
EditCommand	PageIndexChanged	
Init	PreRender	

You can find a number of code examples that utilize the DataGrid control in Chapter 11.

DataList Class

Member of System.Web.UI.WebControls.

The DataList class enables you to display data in a list. The control is similar to the DataGrid control, but instead of displaying multiple categories of data in a tabular (row-and-column) format, the DataList displays a single list of data in a single row. This row can wrap into multiple columns, however.

The various elements of the DataList (header, footer, and items) are divided up into sections and formatted according to templates. Templates are XML sections embedded in the script declaration of the control.

This document is created with the unregistered version of CHM2PDF Pilot

Properties		
AccessKey	EditItemTemplate	RepeatColumns
AlternatingItemStyle	Enabled	RepeatDirection
AlternatingItemTemplate	EnableViewState	RepeatLayout
Attributes	ExtractTemplateRows	SelectedIndex
BackColor	Events	SelectedItemStyle
BorderColor	Font	SelectedItemTemplate
BorderStyle	FooterStyle	SeparatorStyle
BorderWidth	FooterTemplate	SeparatorTemplate
CellPadding	ForeColor	ShowFooter
CellSpacing	GridLines	ShowHeader
ChildControlsCreated	HasChildViewState	Site
ClientID	HeaderStyle	Style
Context	HeaderTemplate	TabIndex
Controls	Height	TagKey
ControlStyle	HorizontalAlign	TagName
ControlStyleCreated	ID	TemplateSourceDirectory
CssClass	IsTrackingViewState	ToolTip
DataKeyField	Items	UniqueID
DataKeys	ItemStyle	ViewState
DataMember	ItemTemplate	ViewStateIgnoresCase
DataSource	NamingContainer	Visible
EditItemIndex	Page	Width
EditItemStyle	Parent	
Methods		
AddAttributesToRender	HasControls	OnPreRender
AddParsedSubObject	IsLiteralContent	OnSelectedIndexChanged
ApplyStyle	LoadViewState	OnUnload
ClearChildViewState	MapPathSecure	OnUpdateCommand
CopyBaseAttributes	MemberwiseClone	RaiseBubbleEvent
CreateChildControls	MergeStyle	Render
CreateControlCollection	OnBubbleEvent	RenderBeginTag
CreateControlStyle	OnCancelCommand	RenderChildren
DataBind	OnDataBinding	RenderContents
Dispose	OnDeleteCommand	RenderControl
Equals	OnEditCommand	RenderEndTag
EnsureChildControls	OnInit	ResolveUrl
Finalize	OnItemCommand	SaveViewState
FindControl	OnItemCreated	ToString
GetHashCode	OnItemDataBound	TrackViewState
GetType	OnLoad	
Events		
CancelCommand	Init	PreRender

This document is created with the unregistered version of CHM2PDF Pilot

DataBinding	ItemCommand	SelectedIndexChanged
DeleteCommand	ItemCreated	Unload
Disposed	ItemDataBound	UpdateCommand
EditCommand	Load	

Listing 2.32 provides an example of a DataList control bound to a Hashtable object.

Listing 2.32 Displaying Data in a Hashtable Object in a DataList Server Control

```
<% @Page language="C#" debug="true" %>
<html>
  <head>
    <title>ASP.NET DataList Control</title>
    <script runat="server">

    void Page_Load(Object Sender, EventArgs e)
    {
      if(!IsPostBack)
      {
        Hashtable h  = new Hashtable();
        h.Add ("SF", "San Francisco");
        h.Add ("AZ", "Arizona");
        h.Add ("CO", "Colorado");
        h.Add ("SD", "San Diego");
        h.Add ("LA", "Los Angeles");

        DataList1.DataSource = h;
        DataList1.DataBind();
      }
    }

    </script>
  </head>
  <body>
    <form runat="server">
      <asp:DataList id="DataList1" runat="server" BorderColor="black"
BorderWidth="1" CellPadding="3" Font-Name="Verdana" Font-Size="8pt">
        <HeaderStyle BackColor="#000000" ForeColor="#FFFF99"></HeaderStyle>
        <AlternatingItemStyle BackColor="#FFFF99"></AlternatingItemStyle>
        <HeaderTemplate>
          National League West
        </HeaderTemplate>
        <ItemTemplate>
          <%# DataBinder.Eval(Container.DataItem, "Value") %>
          [<%# DataBinder.Eval(Container.DataItem, "Key") %>]
        </ItemTemplate>
      </asp:DataList>
    </form>
  </body>
</html>
```

Note that, as with all data-bindable objects, you can bind the DataList to a wide variety of objects. We used the Hashtable object in this example for simplicity, but you could bind to an ArrayList, a DataSet, or any other list type.

DropDownList Class

Member of System.Web.UI.WebControls.

This document is created with the unregistered version of CHM2PDF Pilot

The DropDownList class is the server control abstraction of the HTML SELECT. Like most list controls, it can be bound to data.

Properties			
AccessKey	DataMember	SelectedIndex	
Attributes	DataSource	SelectedItem	
AutoPostBack	DataTextField	Site	
BackColor	DataTextFormatString	Style	
BorderColor	DataValueField	TabIndex	
BorderStyle	Enabled	TagKey	
BorderWidth	EnabledViewState	TagName	
ChildControlsCreated	Font	TemplateSourceDirectory	
ClientID	ForeColor	ToolTip	
Context	IsTrackingViewState	UniqueID	
Controls	Items	ViewState	
ControlStyle	NamingContainer	ViewStateIgnoresCase	
ControlStyleCreated	Page	Visible	
CssClass	Parent	Width	
Methods			
AddAttributesToRender	GetType	Render	
AddParsedSubObject	HasControls	RenderBeginTag	
ApplyStyle	IsLiteralContent	RenderChildren	
ClearChildViewState	LoadViewState	RenderContents	
CopyBaseAttributes	MapPathSecure	RenderControl	
CreateChildControls	MemberwiseClone	RenderEndTag	
CreateControlCollection	MergeStyle	ResolveUrl	
CreateControlStyle	OnBubbleEvent	SaveViewState	
DataBind	OnDataBinding	ToString	
Dispose	OnInit	TrackViewState	
Equals	OnLoad		
EnsureChildControls	OnPreRender		
Finalize	OnSelectedIndexChanged		
FindControl	OnUnload		
GetHashCode	RaiseBubbleEvent		
Events			
Init	PreRender	DataBinding	
SelectedIndexChanged	Unload	Disposed	Load

Listing 2.33 shows an example of a DropDownList object that is bound to an ArrayList object.

Listing 2.33 Binding a DropDownList Control to Data Contained in an ArrayList Object

This document is created with the unregistered version of CHM2PDF Pilot

```
<% @Page language="C#" debug="true" %>
<html>
  <head>
   <title>ASP.NET DropDownList Control</title>
   <script runat="server">

    void Page_Load(Object Sender, EventArgs e)
    {
      if(!IsPostBack)
      {
        ArrayList list = new ArrayList();
        list.Add ("San Francisco");
        list.Add ("Arizona");
        list.Add ("Colorado");
        list.Add ("San Diego");
        list.Add ("Los Angeles");

        DropDownList1.DataSource = list;
        DropDownList1.DataBind();
      }
    }

    void Pick_Click(Object Sender, EventArgs e)
    {
      Label1.Text = "You selected " + DropDownList1.SelectedItem.Text;
    }

  </script>
  </head>
  <body>
    <form runat="server">
      <asp:DropDownList id="DropDownList1" runat="server" BorderColor="black"
BorderWidth="1" Font-Name="Verdana" Font-Size="8pt"></asp:DropDownList>
      <asp:button text="Pick" OnClick="Pick_Click" runat="server" ID="Button1"
/><BR>
      <asp:label id="Label1" runat="server" />
    </form>
  </body>
</html>
```

Use the SelectedItem object contained by the DropDownList control to return information about the item selected by the user. The SelectedItem object (an instance of System.Web.UI.WebControls.ListItem) contains a Text property as well as a Value property, enabling you to retrieve both the displayed value and the key associated with the selected value.

In addition to binding the DropDownList object to a list object such as ArrayList, you can also hard-code the list definitions by using <asp:listitem> subelements in the DropDownList definition.

HttpApplication Class

Member of System.Web. This object is typically accessed as the Application object contained in the ASP.NET Page object.

The HttpApplication class provides a way to store information that has application scope. The Application object contains instance objects such as Request and Response objects (instances of the HttpRequest and HttpResponse

This document is created with the unregistered version of CHM2PDF Pilot

classes, respectively) that you can use to access the contents of conventional HTML forms.

Properties			
Application	Request	Site	
Context	Response	User	
Events	Server		
Modules	Session		
Methods			
AddOnAcquireRequestStateAsync	AddOnReleaseRequestStateAsync	GetHashCode	
AddOnAuthenticateRequestAsync	AddOnResolveRequestCacheAsync	GetType	
AddOnAuthorizeRequestAsync	AddOnUpdateRequestCacheAsync	GetVaryByCustomStrin g	
AddOnBeginRequestAsync	CompleteRequest	Init	
AddOnEndRequestAsync	Dispose	MemberwiseClone	
AddOnPostRequestHandlerExecuteAsync	Equals	ToString	
AddOnPreRequestHandlerExecuteAsync	Finalize		

HttpRequest Class

Member of System.Web.UI.

The HttpRequest class represents a request made by a client. It is typically accessed by programmers through the Request object contained in the ASP.NET Page object.

You can use the Request object to retrieve the value of cookies, read HTTP header information generated by a Web request, get information about the browser that made the request, and poll the client for security-related information.

Properties		
AcceptTypes	Files	PhysicalApplicationPath
ApplicationPath	Filter	PhysicalPath
Browser	Form	QueryString
ClientCertificate	Headers	RawUrl
ContentEncoding	HttpMethod	UrlReferrer
ContentLength	InputStream	UserAgent
ContentType	IsAuthenticated	UserHostAddress
Cookies	IsSecureConnection	UserHostName
CurrentExecutionFilePath	Params	UserLanguages
FilePath	PathInfo	
Methods		
BinaryRead	GetType	SaveAs
Equals	MapImageCoordinates	ToString

This document is created with the unregistered version of CHM2PDF Pilot

Finalize	MapPath	
GetHashCode	MemberwiseClone	

HttpResponse Class

Member of System.Web.UI.

The HttpResponse class represents the data sent to a client in reply to a request. This can include the response itself (handled by the Write method) as well as headers and other configuration data (such as page cache expiry and HTTP headers).

Properties		
Buffer	ContentType	Output
BufferOutput	Cookies	OutputStream
Cache	Expires	Status
CacheControl	ExpiresAbsolute	StatusCode
Charset	Filter	StatusDescription
ContentEncoding	IsClientConnected	SuppressContent
Methods		
AddCacheItemDependencies	Clear	GetType
AddCacheItemDependency	ClearContent	MemberwiseClone
AddFileDependencies	ClearHeaders	Pics
AddFileDependency	Close	Redirect
AddHeader	End	RemoveOutputCacheItem
AppendHeader	Equals	ToString
AppendToLog	Finalize	Write
ApplyAppPathModifier	Flush	WriteFile
BinaryWrite	GetHashCode	

HttpServerUtility Class

Member of System.Web.UI.

The HttpServerUtility class provides a variety of utilities for ASP.NET programmers, such as mapping a file request to the file system of the Web server (the MapPath method) and encoding data for use in a URL (the UrlEncode method). It is typically accessed by ASP.NET developers as the Server object contained by the ASP.NET Page object.

Properties		
MachineName	ScriptTimeout	
Methods		

This document is created with the unregistered version of CHM2PDF Pilot

ClearError	GetHashCode	MemberwiseClone
CreateObject	GetLastError	ToString
CreateObjectFromClsid	GetType	Transfer
Equals	HtmlDecode	UrlDecode
Execute	HtmlEncode	UrlEncode
Finalize	MapPath	UrlPathEncode

HttpSessionState Class

Member of System.Web.UI.

The HttpSessionState class is used to store and retrieve Session state in an ASP.NET application. It is typically accessed by ASP.NET developers in the form of the Session object, contained by the ASP.NET Page object.

Properties		
CodePage	IsReadOnly	Mode
Contents	IsSyncronized	SessionID
Count	Item	StaticObjects
IsCookieless	Keys	SyncRoot
IsNewSession	LCID	Timeout
Methods		
Abandon	Finalize	Remove
Add	GetEnumerator	RemoveAll
Clear	GetHashCode	RemoveAt
CopyTo	GetType	ToString
Equals	MemberwiseClone	

Hyperlink Class

Member of System.Web.UI.WebControls.

The Hyperlink class is the ASP.NET server control abstraction of the HTML A element.

Properties		
AccessKey	EnabledViewState	Style
Attributes	Events	TabIndex
BackColor	Font	TagKey
BorderColor	ForeColor	TagName
BorderStyle	HasChildViewState	Target
BorderWidth	Height	TemplateSourceDirectory

This document is created with the unregistered version of CHM2PDF Pilot

ChildControlsCreated	ID	Text
ClientID	ImageUrl	ToolTip
Context	IsTrackingViewState	UniqueID
Controls	NamingContainer	ViewState
ControlStyle	NavigateUrl	ViewStateIgnoresCase
ControlStyleCreated	Page	Visible
CssClass	Parent	Width
Enabled	Site	
Methods		
AddAttributesToRender	GetHashCode	OnPreRender
AddParsedSubObject	GetType	OnUnload
ApplyStyle	HasControls	RaiseBubbleEvent
ClearChildViewState	IsLiteralContent	Render
CopyBaseAttributes	LoadViewState	RenderBeginTag
CreateChildControls	MapPathSecure	RenderChildren
CreateControlCollection	MemberwiseClone	RenderContents
CreateControlStyle	MergeStyle	RenderControl
DataBind	OnBubbleEvent	RenderEndTag
Dispose	OnDataBinding	ResolveUrl
EnsureChildControls	OnInit	SaveViewState
Equal	OnLoad	ToString
Finalize	OnPreRender	TrackViewState
FindControl	OnUnload	

Use the Text property of the Hyperlink control to specify the text the control should display. Use the NavigateUrl property to determine which page to navigate to. As with the HTML target attribute, you can specify the target window to navigate to by assigning a value to the control's Target property; special values such as "_self" and "_new" are recognized by the control.

Image Class

Member of System.Web.UI.WebControls.

The Image class is the ASP.NET server control abstraction of the HTML IMG element.

Properties		
AccessKey	EnsureChildControls	Page
AddAttributesToRender	Events	Parent
AddParsedSubObject	Finalize	RaiseBubbleEvent
AlternateText	Font	Render
Attributes	ForeColor	RenderChildren
BackColor	HasChildViewState	RenderContents

This document is created with the unregistered version of CHM2PDF Pilot

BorderColor	Height	SaveViewState
BorderStyle	ID	Site
Borderwidth	ImageAlign	Style
ChildControlsCreated	ImageUrl	TabIndex
ClearChildViewState	IsLiteralContent	TagKey
ClientID	IsTrackingViewState	TagName
Context	LoadViewState	TemplateSourceDirectory
Controls	MapPathSecure	ToolTip
ControlStyle	MemberwiseClone	TrackViewState
ControlStyleCreated	NamingContainer	UniqueID
CreateChildControls	OnBubbleEvent	ViewState
CreateControlCollection	OnDataBinding	ViewStateIgnoresCase
CreateControlStyle	OnInit	Visible
CssClass	OnLoad	Width
Enabled	OnPreRender	
EnableViewState	OnUnload	
Methods		
ApplyStyle	FindControl	RenderBeginTag
CopyBaseAttributes	GetHashCode	RenderControl
DataBind	GetType	RenderEndTag
Dispose	HasControls	ResolveUrl
Equals	MergeStyle	ToString
Events		
DataBinding	Init	PreRender
Disposed	Load	Unload

Use the ImageUrl property to specify which image to display. To create an image that works like a button, use the ImageButton control instead.

ImageButton Class

Member of System.Web.UI.WebControls.

The ImageButton class is another ASP.NET server control abstraction of the HTML INPUT type="image" element.

Properties		
AccessKey	ControlStyleCreated	Parent
AlternateText	CssClass	Site
Attributes	Enabled	Style
BackColor	EnableViewState	TabIndex
BorderColor	Events	TagKey
BorderStyle	Font	TagName

This document is created with the unregistered version of CHM2PDF Pilot

BorderWidth	ForeColor	TemplateSourceDirectory
CausesValidation	HasChildViewState	ToolTip
ChildControlsCreated	Height	UniqueID
ClientID	ID	ViewState
CommandArgument	ImageAlign	ViewStateIgnoresCase
CommandName	ImageUrl	Visible
Context	IsTrackingViewState	Width
Controls	NamingContainer	
ControlStyle	Page	
Methods		
AddAttributesToRender	GetHashCode	OnPreRender
AddParsedSubObject	GetType	OnUnload
ApplyStyle	HasControls	RaiseBubbleEvent
CopyBaseAttributes	IsLiteralConent	Render
ClearChildViewState	LoadViewState	RenderBeginTag
CreateChildControls	MapPathSecure	RenderChildren
CreateControlCollection	MemberwiseClone	RenderContents
CreateControlStyle	MergeStyle	RenderControl
DataBind	OnBubbleEvent	RenderEndTag
Dispose	OnClick	ResolveUrl
Equals	OnCommand	SaveViewState
EnsureChildControls	OnDataBinding	ToString
Finalize	OnInit	TrackViewState
FindControl	OnLoad	
Events		
Click	Disposed	PreRender
Command	Init	Unload
DataBinding	Load	

Assign a value to the ImageUrl property to specify which graphic to display (just as you would with the Image control). To execute code in response to a user clicking the image, use the control's Click event, the same as you would for a Button control.

Label Class

Member of System.Web.UI.WebControls.

The Label class provides a way to programmatically create a read-only text region on the page. This region is typically rendered in HTML as a SPAN tag.

Properties		
AccessKey	Enabled	Style

This document is created with the unregistered version of CHM2PDF Pilot

Attributes	EnableViewState	TabIndex
BackColor	Events	TagKey
BorderColor	Font	TagName
BorderStyle	ForeColor	Text
BorderWidth	HasChildViewState	TemplateSourceDirectory
ChildControlsCreated	Height	ToolTip
ClientID	ID	UniqueID
Context	IsTrackingViewState	ViewState
Controls	NamingContainer	ViewStateIgnoresCase
ControlStyle	Page	Visible
ControlStyleCreated	Parent	Width
CssClass	Site	
Methods		
AddAttributesToRender	FindControl	OnUnload
AddParsedSubObject	Finalize	RaiseBubbleEvent
ApplyStyle	GetHashCode	Render
ClearChildViewState	GetType	RenderBeginTag
CopyBaseAttributes	HasControls	RenderChildren
ClearChildViewState	IsLiteralContent	RenderContents
CreateChildControls	MapPathSecure	RenderControl
CreateControlCollection	MemberwiseClone	RenderEndTag
CreateControlStyle	MergeStyle	ResolveUrl
DataBind	OnDataBinding	SaveViewState
Dispose	OnInit	ToString
Equals	OnLoad	TrackViewState
EnsureChildControls	OnPreRender	

A number of code examples involving the Label class are provided throughout this book. In nearly every case, the only member you'll typically need to access is the control's Text property.

LinkButton Class

Member of System.Web.UI.WebControls.

The LinkButton class merges the functionality of a hyperlink with the functionality of the Button control.

Properties		
AccessKey	ControlStyleCreated	Site
Attributes	CssClass	Style
BackColor	Enabled	TabIndex
BorderColor	EnableViewState	TagKey
BorderStyle	Events	TagName

This document is created with the unregistered version of CHM2PDF Pilot

BorderWidth	Font	TemplateSourceDirectory
CausesValidation	ForeColor	Text
ChildControlsCreated	HasChildViewState	ToolTip
ClientID	Height	UniqueID
CommandArgument	ID	ViewState
CommandName	IsTrackingViewState	ViewStateIgnoresCase
Context	NamingContainer	Visible
Controls	Page	Width
ControlStyle	Parent	
Methods		
AddAttributesToRender	Finalize	OnLoad
AddParsedSubObject	GetHashCode	OnPreRender
ApplyStyle	GetType	OnUnload
ClearChildViewState	HasControls	RaiseBubbleEvent
CopyBaseAttributes	IsLiteralContent	Render
ClearChildViewState	LoadViewState	RenderBeginTag
CreateChildControls	MapPathSecure	RenderChildren
CreateControlCollection	MemberwiseClone	RenderContents
CreateControlStyle	MergeStyle	RenderControl
DataBind	OnBubbleEvent	RenderEndTag
Dispose	OnClick	ResolveUrl
Equals	OnCommand	SaveViewState
EnsureChildControls	OnDataBinding	ToString
FindControl	OnInit	TrackViewState
Events		
Click	Disposed	PreRender
Command	Init	Unload
DataBinding	Load	

Handle the control's Click event to execute code when the user clicks the control. To navigate to another Web page, use the Hyperlink control instead.

ListBox Class

Member of System.Web.UI.WebControls.

The ListBox class represents the ASP.NET server control abstraction of the HTML SELECT element.

Properties		
AccessKey	DataTextField	SelectedIndex
Attributes	DataTextFormatString	SelectedItem
AutoPostBack	DataValueField	SelectionMode

This document is created with the unregistered version of CHM2PDF Pilot

BackColor	Enabled	Site
BorderColor	EnableViewState	Style
BorderStyle	Events	TabIndex
BorderWidth	Font	TagKey
ChildControlsCreated	ForeColor	TagName
ClientID	HasChildViewState	TemplateSourceDirectory
Context	Height	Text
Controls	ID	ToolTip
ControlStyle	IsTrackingViewState	UniqueID
	Items	
ControlStyleCreated	NamingContainer	ViewState
CssClass	Page	ViewStateIgnoresCase
DataMember	Parent	Visible
DataSource	Rows	Width
Methods		
AddAttributesToRender	Finalize	OnSelectedIndexChanged
AddParsedSubObject	GetHashCode	OnUnload
ApplyStyle	GetType	RaiseBubbleEvent
ClearChildViewState	HasControls	Render
CopyBaseAttributes	IsLiteralContent	RenderBeginTag
ClearChildViewState	LoadViewState	RenderChildren
CreateChildControls	MapPathSecure	RenderContents
CreateControlCollection	MemberwiseClone	RenderControl
CreateControlStyle	MergeStyle	RenderEndTag
DataBind	OnBubbleEvent	ResolveUrl
Dispose	OnDataBinding	SaveViewState
Equals	OnInit	ToString
EnsureChildControls	OnLoad	TrackViewState
FindControl	OnPreRender	

Page Class

The Page class represents a page request. All controls on a page, as well as utility objects such as Request, Response, Server, and Application (familiar to ASP.old developers), are members of the Page object in ASP.NET.

The page class is the base class from which all ASP.NET pages derive. If you create a code-behind class, it must inherit from this object.

Properties		
Application	Cache	ClientID
ClientTarget	Controls	EnableViewState
ErrorPage	ID	IsPostBack

This document is created with the unregistered version of CHM2PDF Pilot

IsValid	NamingContainer	Page
Parent	Request	Response
Server	Session	Site
SmartNavigation	TemplateSourceDirectory	Trace
UniqueID	User	Validators
Visible		
Methods		
DataBind	DesignerInitialize	Dispose
Equals	FindControl	GetHashCode
GetPostBackClientEvent	GetPostBackClientHyperlink	GetPostBackEventReference
GetType	GetTypeHashCode	HasControls
	IsClientScriptBlockRegistered	IsStartupScriptRegistered
LoadControl	LoadTemplate	MapPath
ParseControl	RegisterArrayDeclaration	RegisterClientScriptBlock
RegisterHiddenField	RegisterOnSubmitStatement	RegisterRequiresPostBack
RegisterRequiresRaiseEvent	RegisterStartupScript	RegisterViewStateHandler
RenderControl	ResolveUrl	VerifyRenderingInServerForm
ToString	Validate	
Events		
AbortTransaction	CommitTransaction	DataBinding
Disposed	Error	Init
Load	PreRender	Unload

It's common for ASP.NET pages to handle the Load event of the Page object as a way to perform initialization when the page loads.

Panel Class

Member of System.Web.UI.WebControls.

The Panel class enables developers to group Web form controls. You may do this for cosmetic purposes (for example, to group the controls on a complicated form into subcategories) or to manipulate controls on a form as a unit.

Properties		
AccessKey	Enabled	Style
Attributes	EnableViewState	TabIndex
BackColor	Events	TagKey
BackImageUrl	Font	TagName
BorderColor	ForeColor	TemplateSourceDirectory
BorderStyle	HasChildViewState	ToolTip
BorderWidth	Height	UniqueID

This document is created with the unregistered version of CHM2PDF Pilot

ChildControlsCreated	HorizontalAlign	ViewState
ClientID	ID	ViewStateIgnoresCase
Context	IsTrackingViewState	Visible
Controls	NamingContainer	Width
ControlStyle	Page	Wrap
ControlStyleCreated	Parent	
CssClass	Site	
Methods		
AddAttributesToRender	Finalize	OnSelectedIndexChanged
AddParsedSubObject	GetHashCode	OnUnload
ApplyStyle	GetType	RaiseBubbleEvent
ClearChildViewState	HasControls	Render
CopyBaseAttributes	IsLiteralContent	RenderBeginTag
ClearChildViewState	LoadViewState	RenderChildren
CreateChildControls	MapPathSecure	RenderContents
CreateControlCollection	MemberwiseClone	RenderControl
CreateControlStyle	MergeStyle	RenderEndTag
DataBind	OnBubbleEvent	ResolveUrl
Dispose	OnDataBinding	SaveViewState
Equals	OnInit	ToString
EnsureChildControls	OnLoad	TrackViewState
FindControl	OnPreRender	

RadioButton Class

Member of System.Web.UI.WebControls.

The RadioButton class represents the ASP.NET server control abstraction of the INPUT type radio. Radio buttons are grouped together; only one button in a group may be selected at a time.

Properties		
AccessKey	Enabled	Style
Attributes	EnableViewState	TabIndex
AutoPostBack	Events	TagKey
BackColor	Font	TagName
BorderColor	ForeColor	TemplateSourceDirectory
BorderStyle	GroupName	Text
BorderWidth		TextAlign
Checked	HasChildViewState	ToolTip
ChildControlsCreated	Height	UniqueID
ClientID	ID	ViewState
Context	IsTrackingViewState	ViewStateIgnoresCase

This document is created with the unregistered version of CHM2PDF Pilot

Controls	NamingContainer	Visible
ControlStyle	Page	Width
ControlStyleCreated	Parent	
CssClass	Site	
Methods		
AddAttributesToRender	Finalize	OnPreRender
AddParsedSubObject	GetHashCode	OnUnload
ApplyStyle	GetType	RaiseBubbleEvent
ClearChildViewState	HasControls	Render
CopyBaseAttributes	IsLiteralContent	RenderBeginTag
ClearChildViewState	LoadViewState	RenderChildren
CreateChildControls	MapPathSecure	RenderContents
CreateControlCollection	MemberwiseClone	RenderControl
CreateControlStyle	MergeStyle	RenderEndTag
DataBind	OnBubbleEvent	ResolveUrl
Dispose	OnCheckedChanged	SaveViewState
Equals	OnDataBinding	ToString
EnsureChildControls	OnInit	TrackViewState
FindControl	OnLoad	

Repeater Class

Member of System.Web.UI.WebControls.

You can use the Repeater control to display bound data in a totally customized way. You do this by creating HTML templates (in a manner similar to the DataList control described earlier in this section).

Properties		
AlternatingItemTemplate	FooterTemplate	Parent
ChildControlsCreated	HasChildViewState	SeparatorTemplate
ClientID	HeaderTemplate	Site
Context	ID	TemplateSourceDirectory
Controls	IsTrackingViewState	UniqueID
DataMember	Items	ViewState
DataSource	ItemTemplate	ViewStategnoresCase
EnableViewState	NamingContainer	Visible
Events	Page	
Methods		
	FindControl	OnItemCreated
AddParsedSubObject	Finalize	OnItemDataBound
	GetHashCode	OnLoad
	GetType	OnPreRender

This document is created with the unregistered version of CHM2PDF Pilot

	HasControls	OnUnload
ClearChildViewState	IsLiteralContent	RaiseBubbleEvent
	LoadViewState	Render
	MapPathSecure	RenderChildren
CreateChildControls	MemberwiseClone	RenderControl
CreateControlCollection		ResolveUrl
DataBind	OnBubbleEvent	SaveViewState
Dispose	OnDataBinding	ToString
Equals	OnInit	TrackViewState
EnsureChildControls	OnItemCommand	

Listing 2.34 shows an example of a Repeater control used to display the contents of the ever-popular Hashtable object. Although you could have used another list control such as the DataGrid or DataList to perform the same work, you can see from the code that the Repeater gives you the capability to embed HTML formatting to have more granular control over the formatting of each row.

Listing 2.34 Using the Repeater Control to Build Customized Output of a Hashtable

```
<%@ Page Language="C#" debug="true" %>
<html>
  <head>
   <script runat="server">

    void Page_Load(Object Sender, EventArgs e)
    {
      if(!IsPostBack)
      {
        Hashtable h  = new Hashtable();
        h.Add ("SF", "San Francisco");
        h.Add ("AZ", "Arizona");
        h.Add ("CO", "Colorado");
        h.Add ("SD", "San Diego");
        h.Add ("LA", "Los Angeles");

        Repeater1.DataSource = h;
        Repeater1.DataBind();
      }
    }

  </script>
  </head>
  <body>
    <form runat="server">
      <asp:Repeater id="Repeater1" runat="server">
        <HeaderTemplate>
          <table border="0" cellpadding="5" cellspacing="1" bgcolor="#000000">
            <tr>
              <td bgcolor="#FFFF99"><b>Team</b></td>
              <td bgcolor="#FFFF99"><b>Abbreviation</b></td>
            </tr>
        </HeaderTemplate>
        <ItemTemplate>
          <tr>
            <td bgcolor="#FFFFFF">
              <%# DataBinder.Eval(Container.DataItem, "Value") %>
            </td>
            <td bgcolor="#FFFFFF">
              <%# DataBinder.Eval(Container.DataItem, "Key") %>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            </td>
          </tr>
        </ItemTemplate>
        <FooterTemplate>
          </table>
        </FooterTemplate>
      </asp:Repeater>
    </form>
  </body>
</html>
```

Remember that the Repeater, like all bound controls, can be bound to any list element, not just the Hashtable.

Note, too, that nothing about the Repeater control necessitates outputting data in an HTML table; you can use the Repeater to render data as a comma-delimited list or as a single-column list with line break (BR) elements, for example.

Table Class

Member of System.Web.UI.WebControls.

The Table class is the ASP.NET server control abstraction of the HTML TABLE element.

Properties		
AccessKey	CssClass	Rows
Attributes	Enabled	Site
BackColor	EnableViewState	Style
BackImageUrl	Events	TabIndex
BorderColor	Font	TagKey
BorderStyle	ForeColor	TagName
BorderWidth	GridLines	TemplateSourceDirectory
CellPadding	HasChildViewState	ToolTip
CellSpacing	Height	UniqueID
ChildControlsCreated	HorizontalAlign	ViewState
ClientID	ID	ViewStateIgnoresCase
Context	IsTrackingViewState	Visible
Controls	NamingContainer	Width
ControlStyle	Page	
ControlStyleCreated	Parent	
Methods		
AddAttributesToRender	FindControl	OnItemCreated
AddParsedSubObject	Finalize	OnItemDataBound
ApplyStyle	GetHashCode	OnLoad
ClearChildViewState	GetType	OnPreRender

This document is created with the unregistered version of CHM2PDF Pilot

CopyBaseAttributes	HasControls	OnUnload
ClearChildViewState	IsLiteralContent	RaiseBubbleEvent
AddParsedSubObject	LoadViewState	Render
ClearChildViewState	MapPathSecure	RenderBeginTag
CreateChildControls	MemberwiseClone	RenderEndTag
CreateControlCollection	MergeStyle	RenderChildren
DataBind	OnBubbleEvent	RenderControl
Dispose	OnDataBinding	ResolveUrl
Equals	OnInit	SaveViewState
EnsureChildControls	OnItemCommand	ToString
		TrackViewState

TableCell Class

Member of System.Web.UI.WebControls.

The TableCell class is the ASP.NET server control abstraction of the HTML TD element.

Properties		
AccessKey	Enabled	Site
Attributes	EnableViewState	Style
BackColor	Events	TabIndex
BorderColor	Font	TagKey
BorderStyle	ForeColor	TagName
BorderWidth	HasChildViewState	TemplateSourceDirectory
ChildControlsCreated	Height	Text
ClientID	HorizontalAlign	ToolTip
ColumnSpan	ID	UniqueID
Context	IsTrackingViewState	VerticalAlign
Controls	NamingContainer	ViewState
ControlStyle	Page	ViewStateIgnoresCase
ControlStyleCreated	Parent	Visible
CssClass	RowSpan	Width
Methods		
AddAttributesToRender	FindControl	OnItemCreated
AddParsedSubObject	Finalize	OnItemDataBound
ApplyStyle	GetHashCode	OnLoad
ClearChildViewState	GetType	OnPreRender
CopyBaseAttributes	HasControls	OnUnload
ClearChildViewState	IsLiteralContent	RaiseBubbleEvent
AddParsedSubObject	LoadViewState	Render

This document is created with the unregistered version of CHM2PDF Pilot

ClearChildViewState	MapPathSecure	RenderBeginTag
CreateChildControls	MemberwiseClone	RenderChildren
CreateControlCollection	MergeStyle	RenderControl
DataBind	OnBubbleEvent	RenderEndTag
Dispose	OnDataBinding	ResolveUrl
Equals	OnInit	SaveViewState
EnsureChildControls	OnItemCommand	ToString
		TrackViewState

TableRow Class

Member of System.Web.UI.WebControls.

The Table class is the ASP.NET server control abstraction of the HTML TR element.

Properties		
AccessKey	Enabled	Site
Attributes	EnableViewState	Style
BackColor	Events	TabIndex
BorderColor	Font	TagKey
BorderStyle	ForeColor	TagName
BorderWidth	HasChildViewState	TemplateSourceDirectory
Cells	Height	
ChildControlsCreated	HorizontalAlign	ToolTip
ClientID	ID	UniqueID
Context	IsTrackingViewState	VerticalAlign
Controls	NamingContainer	ViewState
ControlStyle	Page	ViewStateIgnoresCase
ControlStyleCreated	Parent	Visible
CssClass		Width
Methods		
AddAttributesToRender	FindControl	OnItemCreated
AddParsedSubObject	Finalize	OnItemDataBound
ApplyStyle	GetHashCode	OnLoad
ClearChildViewState	GetType	OnPreRender
CopyBaseAttributes	HasControls	OnUnload
ClearChildViewState	IsLiteralContent	RaiseBubbleEvent
AddParsedSubObject	LoadViewState	Render
ClearChildViewState	MapPathSecure	RenderBeginTag
CreateChildControls	MemberwiseClone	RenderChildren
CreateControlCollection	MergeStyle	RenderControl

This document is created with the unregistered version of CHM2PDF Pilot

DataBind	OnBubbleEvent	RenderEndTag
Dispose	OnDataBinding	ResolveUrl
Equals	OnInit	SaveViewState
EnsureChildControls	OnItemCommand	ToString
		TrackViewState

TextBox Class

Member of System.Web.UI.WebControls.

The TextBox class is the ASP.NET server control abstraction of both the HTML INPUT type text box as well as the TEXTAREA element.

Properties		
AccessKey	EnableViewState	Style
Attributes	Events	TabIndex
AutoPostBack	Font	TagKey
BackColor	ForeColor	TagName
BorderColor	HasChildViewState	TemplateSourceDirectory
BorderStyle	Height	Text
BorderWidth		TextMode
ChildControlsCreated	ID	ToolTip
ClientID	IsTrackingViewState	UniqueID
Columns	MaxLength	
Context	NamingContainer	ViewState
Controls	Page	ViewStateIgnoresCase
ControlStyle	Parent	Visible
ControlStyleCreated	ReadOnly	Width
CssClass	Rows	Wrap
Enabled	Site	
Methods		
AddAttributesToRender	FindControl	OnItemCreated
AddParsedSubObject	Finalize	OnItemDataBound
ApplyStyle	GetHashCode	OnLoad
ClearChildViewState	GetType	OnPreRender
CopyBaseAttributes	HasControls	OnUnload
ClearChildViewState	IsLiteralContent	RaiseBubbleEvent
AddParsedSubObject	LoadViewState	Render
ClearChildViewState	MapPathSecure	RenderBeginTag
CreateChildControls	MemberwiseClone	RenderChildren
CreateControlCollection	MergeStyle	RenderControl
DataBind	OnBubbleEvent	RenderEndTag

This document is created with the unregistered version of CHM2PDF Pilot

Dispose	OnDataBinding	ResolveUrl
Equals	OnInit	SaveViewState
EnsureChildControls	OnItemCommand	ToString
		TrackViewState

WebControl Class

Member of System.Web.UI.Webcontrol. Abstract class.

The WebControl class serves as the base class for Web controls.

Properties		
AccessKey	Enabled	Style
Attributes	EnableViewState	TabIndex
BackColor	Events	TagKey
BorderColor	Font	TagName
BorderStyle	ForeColor	TemplateSourceDirectory
BorderWidth	HasChildViewState	ToolTip
ChildControlsCreated	Height	UniqueID
ClientID	ID	ViewState
Context	IsTrackingViewState	ViewStateIgnoresCase
Controls	NamingContainer	Visible
ControlStyle	Page	Width
ControlStyleCreated	Parent	
CssClass	Site	
Methods		
AddAttributesToRender	FindControl	OnUnload
AddParsedSubObject		RaiseBubbleEvent
ApplyStyle	GetHashCode	Render
BuildProfileTree	GetType	RenderBeginTag
ClearChildViewState	HasControls	RenderChildren
CopyBaseAttributes	IsLiteralContent	RenderContents
CreateChildControls	LoadViewState	RenderControl
CreateControlCollection	MemberwiseClone	RenderEndTag
CreateControlStyle	MergeStyle	ResolveUrl
DataBind	OnBubbleEvent	SaveViewState
Dispose	OnDataBinding	
EnsureChildControls	OnInit	ToString
Equals	OnLoad	TrackViewState
Finalize	OnPreRender	
Events		

This document is created with the unregistered version of CHM2PDF Pilot

DataBinding	Init	PreRender
Disposed	Load	Unload

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 3. Debugging ASP.NET Applications

IN THIS CHAPTER

- Tracing Your Web Application's Activity
- Debugging ASP.NET Applications
- Creating Custom Performance Monitors
- Writing to the Windows Event Log
- Reference

Debugging ASP.old applications was generally only slightly less painful than a trip to the dentist. There was a way to debug ASP.old applications, but it was poorly documented and essentially required Visual InterDev and a team of crack technicians, as well as favorable weather conditions and a whole lot of luck to work correctly.

Many ASP.old developers got into the habit of using code like the following as a way of creating breakpoints in their code:

```
Response.Write "DEBUG: Maybe this will work now."
Response.End
```

This is about the least-efficient kind of debugging code you can possibly write. It's the coding equivalent of driving a car off a cliff just to lift up the hood. At the very least, you should have a way of figuring out what's going on in your application without having to stop its execution.

It should come as no surprise, then, that ASP.NET recognized the severe shortcomings in debugging Web applications and came up with a number of compelling solutions. In ASP.NET, you can perform various useful and detailed inspections into the inner workings of your running applications.

This document is created with the unregistered version of CHM2PDF Pilot

Debugging and tracing in ASP.NET applications doesn't require Visual Studio .NET. (This book doesn't assume you have Visual Studio, either.)

We'll begin our exploration of debugging ASP.NET applications with a discussion of tracing and then move on to debugging and other diagnostic services provided by ASP.NET and the .NET framework.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Tracing Your Web Application's Activity

Tracing is a new feature of ASP.NET that enables you to monitor the activity of your application as it runs. Tracing requires three steps:

1.

1. Equipping a page for tracing
1.
2.

2. Turning tracing on
2.
3.

3. Executing your Web application in Trace mode
3.

When you have gone through these three steps, you'll be able to see the results of the execution of each line of code on each page of your ASP.NET application.

Equipping a Page for Tracing

Any ASP.NET pagecan can run in Trace mode. In fact, you technically don't have to explicitly equip a page for tracing to derive benefit from Trace mode. But equipping a page for tracing enables you to insert custom markers in the trace output, so it's common to include them in all but the most trivial ASP.NET pages. Even better, Trace mode can be turned on and off on at the page level or the application level, so you never need to remove the code that equips a page for tracing. Trace code won't affect performance of your application when tracing is turned off, and you'll never have to worry about your embarrassing ad hoc test output making its way to users because you forgot to comment something out.

NOTE

The Trace object used in ASP.NET is an instance of the TraceContext class, found in the System.Web namespace. (This class is different from the Trace class found in the System.Diagnostics namespace; TraceContext is specific to ASP.NET.)

The properties, methods, and events of the TraceContext class are summarized in the reference section at the end of this chapter.

To equip a page for Trace mode, you make calls to the Write method of the Trace object anyplace in your code you want to receive trace notification. For example, you may be debugging a function that does not appear to be called

This document is created with the unregistered version of CHM2PDF Pilot

during the lifetime of the page. By placing a call to Trace.Write somewhere in the body of the function, you can easily determine whether the function is being called.

NOTE

Because the Trace object is created implicitly by the ASP.NET Page object, you don't need to instantiate it yourself.

Listing 3.1 shows an example of a simple page that is equipped for tracing.

Listing 3.1 A Simple Page Equipped for Tracing with Calls to Trace.Write

```
<% @Page language="C#" debug="true" trace="true" %>
<html>
<head>
  <title>ASP.NET DataList Control</title>
</head>
<script runat="server">

  public void Page_Load(Object sender, EventArgs e)
  {
    Trace.Write("Page_Load starting.");
    if (!IsPostBack)
    {
      Trace.Write("IsPostBack is false; creating data source.");
      Hashtable h = new Hashtable();
      h.Add ("SF", "San Francisco");
      h.Add ("AZ", "Arizona");
      h.Add ("CO", "Colorado");
      h.Add ("SD", "San Diego");
      h.Add ("LA", "Los Angeles");
      Trace.Write("Data binding.");
      DataList1.DataSource = h;
      DataList1.DataBind();
    }
    Trace.Write("Page_Load ending.");
  }

</script>
<body>
  <form runat="server">
    <asp:DataList id="DataList1" runat="server"
      BorderColor="black" BorderWidth="1" CellPadding="3"
      Font-Name="Verdana" Font-Size="8pt">
      <HeaderStyle BackColor="#000000" ForeColor="#FFFF99"></HeaderStyle>
      <AlternatingItemStyle BackColor="#FFFF99"></AlternatingItemStyle>
      <HeaderTemplate>
        National League West
      </HeaderTemplate>
      <ItemTemplate>
        <%# DataBinder.Eval(Container.DataItem, "Value") %>
        [<%# DataBinder.Eval(Container.DataItem, "Key") %>]
      </ItemTemplate>
    </asp:DataList>
  </form>
</body>
</html>
```

This document is created with the unregistered version of CHM2PDF Pilot

You may recognize this page as the DataList example from Chapter 2, "Page Framework." (Book authors enjoy recycling their own code as much as any programmers do.) This version of the code includes calls to Trace.Write to indicate the status of the Page_Load event procedure.

You can see the output of this trace simply by navigating to this page in a browser. The normal page code executes and a voluminous amount of trace information is disgorged to the bottom of the page. Under the heading Trace Information, you should be able to see a number of page-generated trace items (such as Begin Init and End Init) as well as the page's own custom trace custom trace messages (such as Page_Load starting).

Categorizing Trace Output

You can assign a category to the trace output generated by your code. Categorizing trace output can make it easier to sort out trace messages; it's particularly useful when you view output in SortByCategory mode (described in the next section).

You assign a category to a trace message by using an overloaded version of the Trace.Write method. Listing 3.2 shows an example of this.

Listing 3.2 Creating Categorized Trace.Write Output

```
public void Page_Load(Object sender, EventArgs e)
{
    Trace.Write("My Application", "Page_Load starting.");
    if (!IsPostBack)
    {
      Trace.Write("My Application", "IsPostBack is false;" +
            "creating data source.");
      Hashtable h = new Hashtable();
      h.Add ("SF", "San Francisco");
      h.Add ("AZ", "Arizona");
      h.Add ("CO", "Colorado");
      h.Add ("SD", "San Diego");
      h.Add ("LA", "Los Angeles");
      Trace.Write("Data binding.");
      DataList1.DataSource = h;
      DataList1.DataBind();
    }
    Trace.Write("My Application", "Page_Load ending.");
}
```

This is a slightly altered version of the Page_Load event procedure from the previous code example. The only difference is in the pair of strings passed to Trace.Write. When using this form of the method, the first string becomes the category and the second string is the trace message. You can view the trace category alongside the other trace information by viewing the page in Trace mode, as described in the next section.

Enabling Tracing for a Page

You can turn tracing on for a particular page by using an @Page directive. To do this, set the Trace attribute in the @Page directive to true.

This document is created with the unregistered version of CHM2PDF Pilot

```
<@ Page language='C#' trace="true" %>
```

Two Trace modes specify how trace output is sorted—by time or by category.

You control the Trace mode by using the TraceMode attribute in the @Page directive. To sort Trace mode information by category, set the TraceMode attribute to SortByCategory. The default setting, SortByTime, sorts the trace output by time, oldest to newest.

When tracing is activated at the page level, a wealth of information is displayed at the bottom of the normal page output. (Depending on what's normally supposed to be displayed on the page, you may have to scroll down to see the trace information.)

Trace information is divided into the following categories:

-
- Request details—This includes the session ID assigned to the user's session by ASP.NET, the time the request was made, the encoding used in the request and response, the HTTP type, and the HTTP status code.
-
-
- Trace information—This includes trace information automatically generated by ASP.NET, as well as custom trace items generated by calls to Trace.Write from your code. Included in this information is a measurement of how long each operation took to complete. You can use this information to determine where performance bottlenecks exist in the execution of your page.
-
-
- A control tree—This is a hierarchical display of all the controls on the page.
-
-
- A list of cookies transferred by the request—Unless you have cookie-based sessions turned off in your application, typically at least one cookie will be transferred per request (the cookie used to identify the user's session).
-
-
- HTTP headers—These are sent by the server to the browser.
-
-
- Query string values—Values requested by the browser.
-
-

This document is created with the unregistered version of CHM2PDF Pilot

- HTTP server variables—The list of all HTTP server variables sent by the server to the browser.
-

Page-based tracing is useful for performance and debugging purposes. But if you're interested in seeing aggregated tracing information—perhaps to determine how multiple users are accessing elements of an entire Web application—you must use application-level tracing, as described in the next section.

Enabling Tracing in an Application

You can turn tracing on for all the pages in a Web application. To do this, you must make a change in Web.config. Listing 3.3 shows an example of a Web.config settings file that activates tracing.

Listing 3.3 Using the Web.config File to Activate Tracing for an Entire Web Directory

```
<configuration>
   <system.web>
      <trace enabled="true"
         requestLimit="15"
         pageOutput="true"
         localOnly="true" />
   <system.web>
</configuration>
```

In addition to the enabled and pageOutput settings, you can see that the trace configuration settings in Web.config contain a few options that aren't available in the debug settings found in the @Page directive. Specifically, the requestLimit attribute enables you to limit the number of trace requests stored on the server. This option is meaningful when you view aggregate trace information from a remote browser window, as described in the next section.

The localOnly attribute ensures that trace information can be viewed only by users logged on to the Web server machine directly. This prevents remote users from seeing trace output.

For more information on how Web.config works, see Chapter 5, "Configuration and Deployment."

Using Application Tracing from a Remote Browser Window

When application-level tracing is activated, you can view aggregate trace data from a separate browser window. This gives you an aggregate view of all trace information generated by your Web application.

To do this, first equip the application for tracing by adjusting the appropriate settings in Web.config (as described in the previous section).

Next, open two browser windows: one to view a page equipped for tracing in the application; the second to display trace output. (We'll call this second window the trace window.)

In the trace window, navigate to the HTTP handler trace.axd located in the application directory. For example, if

This document is created with the unregistered version of CHM2PDF Pilot

your application is located at http://localhost/myapp/, the Trace mode URL would be http://localhost/myapp/trace.axd. You should be able to see a list of application requests. The list may or may not have any data in it, depending on whether you've refreshed the browser that displays the application page since you started the trace.

After refreshing the application browser a few times, refresh the trace window. You should be able to see a list of trace information. If you navigate to another page in the application and then refresh the trace window, you'll be able to see trace information for that page, too.

You can see that the trace window displays only aggregate information. Further, the number of requests displayed in the window is limited to the number you specified in the Web.config trace setting for the application. You can drill down on each row of information by clicking the View Details link; this displays the same detailed information you see when viewing a single page in Trace mode.

NOTE

Trace.axd isn't a file; instead, it's a link to an ASP.NET feature known as an HTTP handler. You can use the .NET framework to create your own HTTP handlers; this is discussed in Chapter 8, "HttpHandlers and HttpModules."

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Debugging ASP.NET Applications

The large number of ASP.old programmers migrating their code to ASP.NET ensures that error messages are one of the first things an ASP.NET programmer will typically see. Fortunately, not only is the debugging information provided by ASP.NET much better than ASP.old, but you have much more granular control over how debugging information is displayed in ASP.NET.

An ASP.NET debug page is displayed when an unhandled error is encountered, as long as the appropriate @Page or Web.config settings exist. The output of the page is composed of:

- The application (or Web directory) where the error took place
- The type of error and a description of the error
- Details of the error from the compiler
- Source code displaying where the error took place in your code
- The file in which the error took place
- Links to detailed compiler output and complete source code
- Version information for the .NET runtime and ASP.NET

This rich debugging information isn't turned on by default, however. To access rich debugging information, you must turn debugging on, either at the page level (using an attribute of the @Page directive) or for the entire application (using a setting in Web.config).

Enabling Debugging at the Page Level

You can enable debugging at the page level by using an @Page directive.

This document is created with the unregistered version of CHM2PDF Pilot

```
<@ Page language="C#" debug="True" %>
```

Don't try testing this by creating an incorrectly coded ASP.NET page and trying to load it on localhost. Errors on the local machine always create debug information. The debug settings control what users on remote machines see. To test debug settings, you'll need to access your error page on a machine other than the Web server machine.

In case you haven't created one of your own already, Listing 3.4 contains an ASP.NET page that contains a bona fide error. You can use this page to test the way ASP.NET generates debug output code.

Listing 3.4 The First Mistake That Most ASP.NET Programmers Generally Make

```
<% @Page language="C#" debug="false" %>
<HTML>
  <HEAD>
    <TITLE>ASP.NET Error Page</TITLE>
    <SCRIPT runat='server'>
      public void Page_Load(Object Sender, EventArgs e)
      {
        Response.Write "This code will not work."
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    This is my page. There are many others like it, but this one is mine.
  </BODY>
</HTML>
```

Note that this page has its Debug mode set to false. When you first load it (from a remote machine, remember), you'll get a message indicating that something is wrong with the code on the page. But the output won't display any additional information about the error; in particular, the source code display is suppressed. This is an extremely helpful security feature that ensures that your source code won't be displayed in an error condition unless you specifically permit it through an @Page attribute setting or a debug setting in Web.config.

Enabling Debugging at the Application Level

Turning debugging on and off at the page level is easy enough when you're making changes to only a few pages. But early in the development process, when nothing works, you may want to be able to turn Debug mode on for every page in your application.

You can turn debugging on for all the pages in a given folder by changing a setting in the Web.config file in that folder. You can activate Debug mode at the application level by setting the Debug attribute of the compilation section in Web.config.

Listing 3.5 shows an example of a Web.config file that activates debugging for an entire application.

Listing 3.5 A Web.config File That Activates Debugging at the Application Level

```
<configuration>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
   <system.web>
      <customErrors mode="Off" />
      <compilation defaultLanguage="C#"
         debug="true"
         numRecompilesBeforeAppRestart="15">
      </compilation>
   </system.web>
</configuration>
```

That's it. Remember that when you test this option, it's meaningful only when you're debugging the application from a remote machine. Debug mode is always on when you're logging in to the Web application from localhost.

Using the Debug Object

The .NET framework provides a Debug object that you can use to assist with debugging your application. The Debug object is a member of System.Diagnostics.Debug. An instance of this object is created for you automatically, so you shouldn't ever need to instantiate it yourself. If you used the Debug object in Visual Basic, the .NET framework Debug object should be familiar to you.

NOTE

A complete reference to the properties and methods of the Debug object appears at the reference section at the end of this chapter.

The Write method is the most common member of Debug you'll use in your day-to-day programming. By using Debug.Write, you can send information to the development environment you run your ASP.NET application in. If you're using Visual Studio, the strings you send to Debug.Write are displayed in the Output window. (You can view them by running your Web application and then selecting View, Other Windows, Output, or by using the keyboard shortcut Ctrl+Alt+O.)

Listing 3.6 shows a typical call to Debug.Write, used to indicate that an ASP.NET page has loaded and its Page_Load event procedure has been entered.

Listing 3.6 Placing a Call to Debug.Write to Display Debug Information in the Development Environment

```
private void Page_Load(Object Sender , EventArgs e)
{
  Debug.Write("Application initializing. Poot.");
}
```

If you're not using Visual Studio or another integrated development environment to run and debug your ASP.NET application, calls to Debug.Write will go nowhere. A development environment must be present for the output generated by Debug.Write to be meaningful; if you need to send real-time debug information while your application is running, consider using Trace.Write (described earlier in this chapter) or a custom performance monitor (described in the next section).

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Creating Custom Performance Monitors

A performance monitor is a feature of the Windows NT/2000 operating system used to determine operating system resource consumption in real-time. In Windows, performance monitors track such statistics as the amount of processing time being consumed on the system, how much memory is free, and how many times the hard disk is accessed. Software subsystems that run on the server can have their own performance monitors, as well: ASP.NET and SQL Server have their own set of performance monitors that broadcast meaningful information to the operating system about resources consumed by these services.

Any .NET application can be equipped with performance monitors that you create yourself, called custom performance monitors. To expose performance information to the operating system, an application must be equipped to do so.

Running the Windows Performance Monitor Utility

You may have used the Perfmon tool to view the status of various performance monitors on a Windows NT or 2000 server. This tool is found in Windows 2000 under Programs, Administrative Tools, Performance. If you've never used Perfmon before, you can see how it works by running it and then right-clicking its main window and selecting Add Counters from the pop-up menu.

After you do this, a Counters dialog box appears, containing lists of every performance monitor–equipped application installed on your machine. We'll assume you have ASP.NET installed on your computer, so select the ASP.NET Applications category from the list labeled Performance Object. Then select the Anonymous Requests/Sec counter from the list of counters. Finally, click Add to add this counter to your view.

You should notice that the chart begins ticking off the number of requests, which should hover close to zero unless you happen to be performing these experiments on a production Web server (if so—shame, shame). To generate some activity for this performance monitor, simply launch a browser and navigate to an ASPX page (not an ASP page) on localhost. Reload the page in the browser a few times by pressing Ctrl+F5, and then flip back to the Performance window. You should be able to see a modest spike in the number of requests per second.

Creating Performance Monitor Categories

The first step in creating a custom performance monitor is to create your own performance monitor category. Performance monitor categories appear in Perfmon's Performance Object drop-down list; they exist as a way to organize the many performance monitors that can exist on a server.

NOTE

The system provides a number of performance monitor categories, such as Memory, Processor, and so forth. You're not allowed to create performance monitors in these categories. To create a custom performance monitor, you must

This document is created with the unregistered version of CHM2PDF Pilot

create your own performance monitor category first.

To create a new performance monitor category, call the Create method of the PerformanceCounterCategory object. This object is a member of System.Diagnostics, so you may want to import that namespace into your application to work with it.

PerformanceCounterCategory.Create is a static method. The method call enables you to create a performance category and a performance object at the same time. This initializes everything you need to begin performance monitoring with only a few lines of code.

Listing 3.7 shows a procedure that creates a performance category called My Category and an associated performance monitor in that category called My Counter.

Listing 3.7 Initializing a New Performance Monitor Category and Performance Monitor Object

```
public void btnCreateCategory_Click(Object Sender, EventArgs e)
{
  PerformanceCounterCategory.Create ("My Category", "A category just for me",
                "My Counter","A counter just for me");
  Label1.Text = "Performance category created.";
}
```

This form of the PerformanceCounterCategory.Create method takes the following four strings as parameters:

-
- The name of the new category you want to create
-
-
- A text string that describes the category
-
-
- The name of the new performance monitor you want to create
-
-
- A text string that describes the new performance monitor object
-

After you run this code, your new performance monitor category and performance monitor object are created. Note that if you attempt to run this code twice, you'll get an exception. Also, if you're running Perfmon in the background when this code is executed, you'll need to shut down and restart Perfmon to get it to recognize the new category and performance monitor.

Sending Information to a Performance Monitor from Your Application

This document is created with the unregistered version of CHM2PDF Pilot

To provide information about your application to performance monitors, you first create an instance of the PerformanceCounter object.

NOTE

The properties, methods, and events of the PerformanceCounter class are summarized in the reference section at the end of this chapter.

Performance counters always represent some kind of integer count related to the performance of the operation being monitored. After you've created a PerformanceCounter object, you have a choice about how you want to send information to it. You can increment and decrement the integer count using the Increment and Decrement methods of the PerformanceCounter object.

Listing 3.8 shows an example of incrementing the performance counter in response to a button click.

Listing 3.8 Incrementing a Performance Monitor Using a PerformanceCounter Object

```
void Increment_Click(System.Object Sender, System.EventArgs e)
{
   PerformanceCounter pc = new PerformanceCounter();
   pc.CategoryName = "My Category";
   pc.CounterName = "My Counter";
   pc.ReadOnly = false;
   pc.Increment();
}
```

Remember that you must create the performance monitor and its category (described in the previous section) before you can run this code. You can see how this code works by running this code while Perfmon is running in the background. If you have set up a counter for this performance monitor, you should be able to see the graph increase each time you click the button.

In addition to incrementing and decrementing the performance counter object, you can also increase or decrease the value of the counter using the IncrementBy or DecrementBy methods. You can also get or set the raw value of the performance counter by using the RawValue property of the PerformanceCounter object.

Deleting Performance Monitor Categories

You can delete a performance monitor category by executing the Delete method of the PerformanceCounterCategory object. Listing 3.9 shows an example.

Listing 3.9 Deleting a Performance Category

```
public void btnDeleteCategory_Click(Object Sender, EventArgs e)
{
   PerformanceCounterCategory.Delete("My Category");
```

This document is created with the unregistered version of CHM2PDF Pilot

```
}
```

Deleting a performance category deletes all performance monitors associated with that category. If Perfmon is running when you delete a performance category, it will continue monitoring that category; you must shut down and restart Perfmon to see the change.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Writing to the Windows Event Log

It's common for server software to persistently record information about what it's doing. This activity is called logging. It's common for programmers to spend time creating logging features in the applications they write—both as a debugging tool and as a way for users and system administrators to see what's going on with the software. However, Windows programmers typically don't create their own logging facilities in the applications they create, because the operating system provides one for them: the Windows event log.

The Windows event log is a central, system-managed place where any application, service, or device can log information. It is available on Windows NT and Windows 2000. Logged events usually contain status information or failure messages. You are free to use the event log from any of your applications, including ASP.NET applications, as a way of persistently storing information pertaining to your application's behavior in a way that's easy for system administrators to access.

NOTE

Logging serves a similar function to performance monitoring (discussed earlier in this chapter). But it differs in the area of how the data is stored. Monitoring is a real-time view of what your application is doing at any given moment. Monitoring information is stored in memory and does not normally survive a system reboot. Logging, on the other hand, is persisted to disk and provides a historical record of your server application's behavior over a long period of time.

In Windows 2000 Server, you can access the event log through the Event Viewer application found in the Administrative Tools group under the Start menu. Events are divided into three categories: application, security, and system. Your application can read from any log and write information to the application log, as described in the next section.

Using the EventLog Class

Any application, including ASP.NET applications, can access the event log. Your applications will write information to the application log (rather than to the security or system logs).

The .NET framework provides a class for handling the event log. This is the EventLog class, found in System.Diagnostics.

NOTE

The properties, methods, and events of the EventLog class are summarized in the reference section at the end of this chapter.

This document is created with the unregistered version of CHM2PDF Pilot

To send information to the application log, you use the EventLog class. This class is created for you automatically, so you do not need to instantiate it (although you can if you need to).

To perform logging, you first use the EventLog object to create an event source. An event source is a way of identifying where a log entry came from. Many data sources can write to the same log; for example, many applications typically write error messages to the application log, and a variety of server processes create entries in the system log.

Event sources must be registered before you write information to the log. You do this by calling the CreateEventSource method of the EventLog object.

NOTE

For applications you create, unless you have a really good reason to create your own log, it's probably best to toss your own logs into the application log with everything else. That's where system administrators will look for it.

Only the first eight characters of a log name are significant. This means you can't create a log named, for instance, Application Data, because Application is a built-in log created by Windows.

Listing 3.10 shows an example of how to write information to the Windows event log. This code first checks to see if an event source called MyApp exists; if not, the code creates it. The code then sends information to the event log by calling the WriteEntry method of the EventLog object.

Listing 3.10 Writing an Event to the Windows Event Log

```
void  Button1_Click(Object Sender, EventArgs e)
{
  if(!EventLog.SourceExists("MyApp"))
  {
    EventLog.CreateEventSource("MyApp", "Application");
  }
  EventLog.WriteEntry("MyApp", "This is just a test.",
            EventLogEntryType.Information);
}
```

The WriteEntry method is overloaded; the code example shows the most commonly used form. The first parameter is a string representing the event source. The second parameter is the message to insert in the log. The third parameter is an event type; this is a member of the enumeration System.Diagnostics.EventLogEntryType.

You can view the output of this code by launching the Event Viewer (found in Programs, Administrative Tools). After running the code, click Application Log. You should be able to see an Information entry for the MyApp event source. Double-clicking the event displays a property sheet that shows you the detail for the event (the description

This document is created with the unregistered version of CHM2PDF Pilot

"This is just a test").

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Reference

This section provides a quick reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book. For the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

This chapter covers the debugging, performance-monitoring, and event-logging classes in ASP.NET. The following sections provide a brief reference to the properties, methods, and events provided by those classes.

Debug Class

Member of System.Diagnostics.

The Debug object gives developers access to a number of useful tools to manage the debugging process.

It is not necessary to instantiate a Debug object; an instance of the object is always available in an ASP.NET page.

Properties		
AutoFlush	IndentSize	
IndentLevel	Listeners	
Methods		
Assert	Indent	WriteLine
Close	Unindent	WriteLineIf
Fail	Write	
Flush	WriteIf	

EventLog Class

Member of System.Diagnostics.

The EventLog class is used to read and write information from the Windows event log. This is used primarily to record diagnostic information pertaining to an application, particularly failure information.

Properties		
Container	Events	Site
DesignMode	Log	Source

This document is created with the unregistered version of CHM2PDF Pilot

EnableRaisingEvents	LogDisplayName	SynchronizingObject
Entries	MachineName	
Methods		
BeginInit	EndInit	InitializeLifetimeService
Clear	Equals	LogNameFromSourceName
Close	Exists	MemberwiseClone
CreateEventSource	Finalize	SourceExists
CreateObjRef	GetEventLogs	ToString
Delete	GetHashCode	WriteEntry
DeleteEventSource	GetLifetimeService	
Dispose	GetService	
	GetType	
Events		
Disposed	EntryWritten	

PerformanceCounter Class

Member of System.Diagnostics.

You create an instance of this class to expose your application's performance information to performance monitors.

Properties		
CategoryName	CounterType	MachineName
Container	DesignMode	RawValue
CounterHelp	Events	ReadOnly
CounterName	InstanceName	Site
Methods		
BeginInit	Equals	InitializeLifetimeService
Close	Finalize	MemberwiseClone
CloseSharedResource	GetHashCode	NextSample
CreateObjRef	GetLifetimeService	NextValue
Decrement	GetService	RemoveInstance
Dispose	GetType	ToString
	Increment	
EndInit	IncrementBy	
Events		
Disposed		

TraceContext Class

Member of System.Web.

This document is created with the unregistered version of CHM2PDF Pilot

TraceContext provides Trace mode functionality in ASP.NET Web application development. It is typically accessed as the Trace object contained by the ASP.NET Page object.

Properties		
IsEnabled	TraceMode	
Methods		
Equals	GetType	Warn
Finalize	MemberwiseClone	Write
GetHashCode	ToString	

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 4. State Management and Caching

IN THIS CHAPTER

- State Management: What's the Big Deal?
- Caching
- Class Reference

ASP.NET introduces several new options for managing session state and introduces a whole new category of functionality, caching. This chapter covers all of the new session state options and introduces the new caching options.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

State Management: What's the Big Deal?

HTTP by its very nature is a stateless protocol. This doesn't mean it disregards geographic boundaries; it means that it is not connection oriented. No request of the Web server can rely on data supplied by some other request. To understand this concept, let's look at an example of how a browser requests a Web page.

When a user types in the address of a Web site, www.deeptraining.com/default.aspx, for example, the Web browser performs a number of steps prior to displaying the page. First, the Web browser converts the hostname, in this case www, to an IP address. It does this by querying a DNS server and asking for the IP address. In our sample, this brings back 192.168.1.200. Next, the Web browser opens a TCP socket to the Web server using port 80. After the connection is made, the Web browser sends a GET /default.asp command. The Web server streams the HTML contents of the page back to the browser. The Web server then closes the TCP socket connection to the Web browser.

NOTE

HTTP 1.1 allows more than one command to be sent without closing the socket connection. This is called Keep-Alive. However, each command stands on its own and should not rely on any state from previous commands.

This series of events is visibly demonstrated by using Telnet instead of a Web browser to communicate with a Web server. Listing 4.1 shows what this would look like.

Listing 4.1 A Sample HTTP Request Using Telnet

```
GET /DEFAULT.ASPX
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 28 Mar 2001 00:38:29 GMT
Set-Cookie: ASP.NET_SessionId=sfdaa145jb0mdv55nnhgic55; path=/
Cache-Control: private
Content-Type: text/html; charset=iso-8859-1
Content-Length: 130

<html>
<head>
<title>Welcome to ASP.NET</title>
<body>
Welcome to ASP.NET!
<img src="/image1.jpg"><img src="/image2.jpg">
</body>
</html>

Connection to host lost.
```

To replicate this example in Windows 2000, open up a command prompt and type the following:

This document is created with the unregistered version of CHM2PDF Pilot

```
TELNET localhost 80
```

This will open up the Telnet connection to your Web server. Now you need to request a page. If you have the default Internet Information Server installation, you can do a GET /localstart.asp to retrieve the Start page. You must type the command exactly; while in Telnet, the Backspace key doesn't work to correct mistakes.

The Web browser receives the HTML, parses it, and is then ready to receive additional requests. In this example, two more requests would need to be made: one for image1.jpg and another for image2.jpg. Each request causes another TCP socket connection to be made over port 80 and then a GET /image1.jpg command to be sent. The Web server streams back the image and then closes the port. A diagram of this process is shown in Figure 4.1.

Figure 4.1. Steps in a standard browser request.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Caching

Do you have a page on your Web site that is built from the database, but the data changes only once a day? Perhaps you have a busy order page that must populate a list of states and countries out of the database several thousand times a day. How often do you need to add a new state? Clearly, these sorts of updates happen very infrequently. Are you hitting the limits of how far you can scale your database server? Are you looking for ways to reduce the load instead of spending $100,000 to upgrade the server? These are the types of scenarios where it makes sense to take a hard look at caching.

Output Caching: Caching an Entire Page

The simplest type of caching in ASP.NET is called output caching. Output caching takes the results from an ASP.NET page request and stores it in memory. Future requests for the same page are then served out of memory instead of being served by re-creating the page from scratch. (See Figure 4.6.) This can yield enormous savings in processor utilization on both the Web server and the database server, depending on the work the page performs. Page load times are decreased because after the first request, the page behaves as though it is a static HTML page.

Figure 4.6. Page requests are intercepted and served from cache.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Class Reference

This section provides a quick interface reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book, so for the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

HttpSessionState Class

Member of System.Web.SessionState.

Assembly: System.Web.dll.

The HttpSessionState class is instantiated and placed inside the Session property of the Page object.

Properties		
CodePage	Contents	Count
IsCookieless	IsNewSession	IsReadOnly
IsSynchronized	Item	Keys
LCID	Mode	SessionID
StaticObjects	SyncRoot	TimeOut
Methods		
Abandon	Add	Clear
CopyTo	GetEnumerator	Remove
RemoveAll	RemoveAt	

HttpCachePolicy

Member of System.Web.

Assembly: System.Web.dll.

The HttpCachePolicy class allows you to control many of the parameters that the <%@ OutputCache %> directive controls. It also adds several more methods to give additional control over caching.

Properties		
VaryByHeaders	VaryByParams	

This document is created with the unregistered version of CHM2PDF Pilot

Methods		
AddValidationCallback	AppendCacheExtension	SetCacheability
SetETag	SetETagFromFile Dependencies	SetExpires
SetLastModified	SetLastModifiedFrom FileDependencies	SetMaxAge
SetNoServerCaching	SetNoStore	SetNoTransforms
SetProxyMagAge	SetRevalidation	SetSliding Expiration
SetValidUntilExpires	SetVaryByCustom	

HttpCacheVaryByHeaders

Member of System.Web.

Assembly: System.Web.dll.

The HttpCacheVaryByHeaders class provides a type-safe way to set the VaryByHeaders property of the HttpCachePolicy class for a number of well-known headers.

Properties		
AcceptTypes	Item	UserAgent
UserCharSet	UserLanguage	
Methods		
VaryByUnspecifiedParameters		

HttpCacheVaryByParams

Member of System.Web.

Assembly: System.Web.dll.

This document is created with the unregistered version of CHM2PDF Pilot

The HttpCacheVaryByParams class provides a type safe way to set the VaryByParams property of the HttpCachePolicy class.

Properties	
IgnoreParams	Item

Cache

Member of System.Web.Caching.

Assembly: System.Web.dll.

The cache class is a store for caching frequently used resources on the server.

Fields		
NoAbsoluteExpiration	NoSlidingExpiration	
Properties		
Count	Item	
Methods		
Add	Get	GetEnumerator
Insert	Remove	

CacheDependency

Member of System.Web.Caching.

Assembly: System.Web.dll.

Keeps track of dependencies for items in the cache. This class cannot be inherited from.

Constructors

```
public CacheDependency(string);
public CacheDependency(string[]);
public CacheDependency(string, DateTime);
public CacheDependency(string[], DateTime);
public CacheDependency(string[], string[]);
public CacheDependency(string[], string[], CacheDependency);
public CacheDependency(string[], string[], DateTime);
public CacheDependency(string[], string[], CacheDependency, DateTime);
```

This document is created with the unregistered version of CHM2PDF Pilot

Properties
HasChanged
Methods
Dispose

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 5. Configuration and Deployment

IN THIS CHAPTER

- Understanding Configuration Files
- Global and Local Configuration Files
- Structure of Configuration Files
- Accessing Configuration Files Programmatically
- Editing Web Configuration Files in Visual Studio .NET
- Initializing Web Applications Using Global.asax
- Using XCOPY for Deployment
- Managing the Global Assembly Cache

Deploying applications under ASP.old was fairly simple—most of the time. You designated a folder as scriptable under Internet Services Manager, copied your script files into that folder, and requested the ASP pages through a Web browser. If something went wrong, you got a 404 Not Found error, which sent you back either to Windows Explorer to locate the missing file or into Internet Services Manager to change an incorrect setting. On paper, it all looked pretty simple.

Under this old model, the trouble came when your ASP application depended on external resources to run. For example, if your application needed to periodically retrieve or store information from the system registry, or (even worse) if your application depended on one or more COM components, you found yourself in a situation in which you could not easily and automatically replicate your Web application on another server. This meant that for all but

This document is created with the unregistered version of CHM2PDF Pilot

the most trivial ASP.old applications, it was a pain to move your application from a development server to a production server. The problems involved in replicating external dependencies got much worse in situations in which you were required to deploy your application to a series of identical servers (such as a Web farm).

ASP.NET promises to make the process of deploying your Web applications much easier, no matter what kind of application architecture or server you're working with. It does this by doing away with certain dependencies (such as the system registry and the IIS metabase) and minimizing the impact of others—most notably, it got rid of the requirement that a precompiled component be registered, as is the case with COM components.

In addition to making deployment simpler, ASP.NET makes the process of configuration much easier, as well. In the past, IIS and ASP configuration files were stored in the registry and were accessible only through the registry editor or (more com-monly) Internet Services Manager. But in many cases, important configuration information would get lost in the GUI of the management console, which changed from one version of Windows to the next.

Storing IIS and ASP configuration data in the registry also meant that configuration itself became a new kind of deployment problem, too, because you couldn't easily provide registry settings for your Web applications to suit multiple machines or multiple customers.

In ASP.NET, many application-level settings are available through XML configuration files that you can view and change using any text editor. This has advantages and disadvantages, as we'll discuss later in this chapter. But by and large, the capability to easily distribute your configuration file along with the application itself is a huge boon to application developers.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Understanding Configuration Files

Most software applications need to maintain bits of information about how the software is supposed to run. Web applications are no different.

In ASP.old, you had a limited number of ways to manage application configuration data. You could

-
- Embed configuration data in the registry and hope that the person who deploys your application can insert the necessary registry settings properly.
-
-
- Use script files such as global.asa or constants located in your own script files.
-
-
- Use custom text files that could be read and written to as needed.
-

All three of these techniques have significant drawbacks. Registry-based configuration is difficult to manage. Changing a script file often requires a programmer, and errors inserted into script files can sabotage an entire application. Custom text files alleviate this problem, but in many cases using an external file means that a reference to that file must be explicitly included in every script file that accesses it.

The designers of ASP.NET recognized that application configuration was never as straightforward as it could have been. In .NET, Web applications have a number of new configuration options and features (which we'll discuss in this chapter and elsewhere in this book). But more importantly, the manner in which you configure your Web application is now totally standard and fairly simple to manage. In ASP.NET, applications are configured with just two XML files—Machine.Config and Web.Config. You can easily make changes to and redistribute these to enforce settings in your applications, as you'll see in the next section.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Global and Local Configuration Files

Servers that run .NET Web applications will typically have multiple ASP.NET configuration files. One such file, known as the machinewide or global configuration file, is named Machine.Config; this file is created for you when you install the .NET framework. The exact location of this file depends on your machine configuration and the version of the .NET framework you're running, but it should be located in \Windows\Microsoft.NET\Framework\[version]\CONFIG. (Replace [version] with whichever version you're actually running.)

Additionally, any Web-accessible folder can optionally contain a Web.Config file that stores settings that are relevant to the ASP.NET scripts in that folder. These settings override the machinewide settings found in Machine.Config.

ASP.NET Web applications can actually have multiple Web.Config files. This can occur when the application contains subfolders. Each subfolder can have its own Web.Config; the configuration settings found in folders that are located deeper in the hierarchy override the settings found at higher levels.

It's not necessary for any of your ASP.NET Web applications to have Web.Config files. If your application has no configuration file, it inherits the settings found in the global configuration file (Machine.Config). When you make a change to a configuration file, ASP.NET automatically picks up the change and applies it. You do not have to restart the server or reboot the machine to get ASP.NET to recognize the changes you made. Also, although the Web.Config file is by definition located in a Web-accessible directory, client browsers are prohibited from downloading Web-configuration files (through an IIS security setting that is set at the time the .NET framework is installed on your system).

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Structure of Configuration Files

As mentioned earlier, ASP.NET configuration files are XML documents. The root node of the document is always called configuration. Within the configuration node are a variety of nodes and subnodes that contain additional settings; these are grouped into section handlers.

The next few sections describe the default section handlers in the System.Web section of a Web configuration file. Because each Web.Config file ultimately inherits its settings from Machine.Config, the descriptions apply to both files.

Authentication Settings

Authentication refers to the process whereby a user is granted or denied access to the page based on security credentials.

ASP.NET supports three authentication modes:

-
- Windows authentication
-
-
- Cookie-based authentication
-
-
- Microsoft Passport authentication
-

In Windows authentication, the user has an account on the Windows NT/Windows 2000 server on which the Web application is located; if the user is not located on the same subnet as an authenticating server, the user must supply a username and password when the browser initially accesses a Web application.

In cookie-based authentication, an encrypted chunk of data is deposited on the user's computer and read by the server each time the user accesses the site.

Passport authentication is similar to cookie-based authentication. It enables users to use the same security credentials to access any one of a number of sites. This is accomplished by storing and authenticating user information in a central location (managed by Microsoft).

NOTE

This document is created with the unregistered version of CHM2PDF Pilot

It is possible to use Passport authentication whether or not your site uses ASP.NET. You can get more information about implementing Passport authentication on your site at http://www.passport.com/business.

To specify one of these authentication schemes, you make a change to the authentication section of the Web.Config or Machine.Config files.

Figure 5.1 shows a typical authentication section and describes some of its settings.

Figure 5.1. Description of the authentication section of the Web configuration file.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Accessing Configuration Files Programmatically

Because the Web configuration files are XML documents, you could use the XML-handling objects (discussed in Chapter 10, "Using XML") to store and retrieve information in Web.Config and Machine.Config. This might be useful when you want to store application-specific information in Web.Config, such as a database connection string.

However, you don't need to go to the trouble of parsing Web.Config to store and retrieve custom values from the file—there is an easier way. To store application-specific settings in Web.Config, simply create a section called appSettings and add the settings you want as key/value pairs in that section.

In the following example, the connection string is given the key "pubsdata"; the value of the setting is the familiar ADO connection string.

```
<configuration>
    <system.web>
        <customErrors mode="Off" />
    </system.web>
    <appSettings>
      <add key="pubsdata" value="SERVER=localhost;DATABASE=pubs;UID=sa;PWD=mypass;"
/>
    </appSettings>
</configuration>
```

Note that you aren't limited to storing database connection strings in appSettings, but appSettings is a reasonable solution to the common problem of where to store connection string data without sticking it into a constant in your code or using a conventional include file.

To retrieve the custom application setting, you use the AppSettings collection contained in the ConfigurationSettings object. This object, a member of the System.Configuration namespace in the .NET framework, enables you to access the values stored in Web.Config by their key. So, for example, to retrieve the pubsdata value specified in the previous example, you'd use the expression

```
Response.Write(ConfigurationSettings.AppSettings["eqguild"]);
```

Note that it's not necessary to create an instance of the ConfigurationSettings object. As with other objects such as Page and Response, a single instance of ConfigurationSettings is always available to code that executes in your ASP.NET page.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Editing Web Configuration Files in Visual Studio .NET

Like any other XML file, you can edit Web configuration files in Visual Studio .NET. In fact, when you create a Web application project in Visual Studio .NET, the system adds a default Web.Config file to your project automatically.

Several advantages exist to editing Web configuration files in Visual Studio .NET. Among these are color-coded syntax and the capability to navigate quickly through the configuration file using the outline mode common to all XML files in the Visual Studio development environment.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Initializing Web Applications Using Global.asax

ASP.old provided a file, global.asa, that provided functionality specific to the ASP application and session. You would typically put initialization code in global.asa, which provided a set of event-handling procedures for this purpose.

ASP.NET provides a similar functionality in the file Global.asax. Four categories of entries can appear in a Global.asax file:

-
- Application directives
-
-
- Code declaration blocks
-
-
- Server-side object tags
-
-
- Server-side includes
-

Each of these four categories of entries is discussed in more detail in the next few sections.

Note that when you make a change to Global.asax, as with all ASP.NET files, the binary representation of the script will be automatically recompiled. This isn't a big deal in the case of most scripts, but in the case of Global.asax, making a change and recompiling has the potentially troublesome side effect of wiping out all existing sessions (including any data stored in session variables). Be aware of this when making changes to Global.asax on a production system; as with any major change to a production site, you may want to use Windows Scheduler to upload the updated version of the file to the production server at a time when traffic on your site is at its lowest, such as the middle of the night or on a weekend. Also, storing session data out-of-process or in SQL Server will prevent this problem from happening.

Note, too, that the code in Global.asax can't be run directly; nothing happens when a user navigates to Global.asax using a browser, for example. This prevents users from viewing or running the code contained in this file.

Application Directives in Global.asax

An application directive is a line of code that instructs ASP.NET to take some special action regarding the execution of the page.

This document is created with the unregistered version of CHM2PDF Pilot

Three application directives are supported in Global.asax:

-
- @Application, which allows you to specify an inheritance relationship between the application object and another object
-
-
- @Assembly, which links an assembly to the application
-
-
- @Import, which imports a namespace
-

You've seen examples of @Import directives in virtually every code example in this book so far, so we won't go into them again here.

Using the @Application Directive in Global.asax

The @Application directive supports two attributes: a class name and a description string.

```
<%@ Application inherits="MyComp.MyApp" [sr]
                description="My Custom Application %>
```

You use the @Application directive in situations where you want to create a custom Application object that inherits from another object.

Note that the options available in the @Application directive in Global.asax are different from the attributes of the @Application directive that are legal in normal ASPX pages.

Using the @Assembly Directive in Global.asax

The @Assembly directive represents a reference to an external assembly (a component) that is not contained in the application's \bin directory. The advantage of referencing an assembly is that you get the performance and type-safety benefits of early binding, as well as IntelliSense when working with the assembly's objects in the Visual Studio development environment.

```
<%@ Assembly Name="PrattleFreeApp.Customer" %>
```

ASP.NET assemblies are typically contained in a \bin subdirectory under the application directory. Such assemblies are automatically referenced by all pages in an ASP.NET application. The @Assembly directive is used in situations where, for some reason, the assembly is stored in an unexpected place.

This document is created with the unregistered version of CHM2PDF Pilot

The syntax for referencing an external assembly in Global.asax is the same as referencing an external assembly in a normal ASP.NET page.

Code Declaration Blocks in Global.asax

Code declaration blocks are events associated with Page objects such as Server and Application. You can write code in event handlers for these objects in Global.asax. The advantage of this is centralized management of initialization code. Because code in Global.asax is guaranteed to execute no matter which page initially uses to access the application, you don't have to worry about including it (or copying it) into every file of your application.

The event procedures that can be included in Global.asax (in the order in which the events fire) are

-
- Application_OnStart
-
-
- Session_OnStart
-
-
- Session_OnEnd
-
-
- Application_OnEnd
-

Listing 5.2 shows an example of an event handler for the Session object.

Listing 5.2 Example of an OnStart Event Handler in Global.asax

```
<SCRIPT runat='server' language='c#'>

  void Session_OnStart()
  {
    Response.Write("Starting a new session!<BR>");
    Session["StartTime"] = DateTime.Now;
    Session.Timeout = 45;
  }

</SCRIPT>
```

You can test this script by dropping it into a Web-accessible directory and then navigating to an ASPX script file located in that directory. Don't try to navigate directly to Global.asax; it won't work. Also, the code in Global.asax isn't accessible from ASP.old scripts, so don't try testing it with an .ASP file.

This code is very straightforward; it sends a message to the browser for debugging purposes, stores the date and time the session began in a Session variable, and then sets the session timeout to 45 minutes. The code in this event

This document is created with the unregistered version of CHM2PDF Pilot

procedure executes only the first time a user navigates to one of the pages in the Web application; a Web application is defined as any ASP.NET Web script in a given directory.

Server-Side Object Tags in Global.asax

In many cases, it's useful for a Web application to have access to an object globally. This enables you to share objects across multiple pages. You can also specify whether the object has application or sessionwide scope. You use server-side object tags in Global.asax to do this, as shown in Listing 5.3.

Listing 5.3 Declaring an Application-Scoped Object in Global.asax

```
<object id="users"
        scope="application"
        class="System.Collections.ArrayList"
        runat="server"/>

<SCRIPT runat='server' language='c#'>

    public void Session_OnStart()
    {
        String usr;
        usr = Request.UserHostAddress;

        users.Add(usr);

        Response.Write("Global.asax: New session created.");

    }

</SCRIPT>
```

In addition to the definition of the application-level object called users, the code includes a Session_OnStart event handler that adds the user's IP address to the user's ArrayList at the time the session is initiated. (In a real application, you would need a corresponding Session_OnEnd code to remove the user's IP address from the array when the session timed out.)

To display the contents of the array, use code similar to that shown in Listing 5.4.

Listing 5.4 Displaying Results of the Active-User Array

```
<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    // NB the variable "users" is defined in global.asax

    Response.Write(users.Count + " users have active sessions.<BR>");

    foreach(String usr in users)
    {
      Response.Write(usr + "<BR>" );
    }

  }
</SCRIPT>
```

This document is created with the unregistered version of CHM2PDF Pilot

This code will work in any script that is contained in the application directory. To test it, you will either have to access the same page from two computers or launch two separate sessions on the same machine by using a different Web browser (such as Netscape or Opera) for your second session.

When you access the Web server from your local machine, the IP address will always be 127.0.0.1.

Note that you don't have to use this technique to use an object globally; instead, you can place objects in the Session object or (better yet) place the object in the ASP.NET cache. For information on this, see Chapter 4, "State Management and Caching."

Server-Side Includes in Global.asax

Server-side includes (SSIs) are the same in Global.asax as they are in any other server-side script. The SSI directive tells the Web server to include an external file in the script; it works the same as if you had copied and pasted it into the script yourself.

The syntax for a server-side include in Global.asax is the same as in any other server-side script in ASP or ASP.NET:

```
<!-- #include File = "MyFile.inc" -->
```

It's typical to see server-side includes used as a code-sharing tactic. Unfortunately, maintaining server-side includes can be unwieldy, particularly in situations where the includes are numerous or contain large amounts of code. Other disadvantages exist to using includes, notably that you don't have access to IntelliSense in the Visual Studio development for procedures contained in SSIs.

Rather than using SSIs, you may want to consider defining your global procedures as objects and referencing them as components instead.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Using XCOPY for Deployment

Microsoft uses a shorthand term for the ease-of-deployment features in ASP.NET—it's called XCOPY deployment. This refers to a command that was brought into Windows from the DOS world. An evolved version of the DOS COPY command, XCOPY adds more powerful features, including the capability to create entire folders and subfolder structures where necessary. In situations in which you use the command to copy a folder and child subfolders, XCOPY can create identical folder structures on the destination disk.

Additionally, XCOPY has the capability to copy only those files that are newer than files on the destination drive. This is a big benefit for large sites that don't want to copy all 10,000 files each time they make a few changes, but it's an even more important feature for developers who frequently make little changes to several files and then forget which files are newer—the ones on the development machine or the ones on the server. You can painstakingly compare the dates and times that each file in your application was last modified, but that's grunt work that the computer should take care of for you. XCOPY deployment performs that grunt work for you.

The ultimate goal of XCOPY deployment is to have an automated way to send changes from your development machine to your test server and on to your production machine when everything is all ready. With that in mind, we'll run through a few scenarios that demonstrate how to use XCOPY in real life. (At the end of this section, you'll find a quick reference to all XCOPY's options in case you ever need to do something more exotic.)

In our scenarios, we'll set up two folders on the same machine, C:\SOURCE and C:\TARGET. The objective in each case will be to copy some files (and, optionally, a directory structure) from one place to another. Figure 5.10 illustrates the state of the file system when we begin.

Figure 5.10. Initial state of file system before using XCOPY.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Managing the Global Assembly Cache

One of the advantages to working with ASP.NET is that code you write is compiled for you seamlessly and transparently. This is a useful feature; however, you may find yourself in a situation where you want to reuse the functionality found in one application in another. You may even want to share software functionality across two radically different types of applications. A Web application and a thick-client application on the Windows desktop, for example, might share a common set of code that performs a set of operations specific to your business. You can accomplish this through the use of components. (Because corporate application developers often use components to share standard business-specific code functionality across multiple applications, components are sometimes also referred to as business logic components.)

A component is a piece of software designed to be used by another piece of software. Windows has had components throughout its history. Initially, components were packaged in the form of procedural dynamic link libraries (DLLs), but for ease of programming and interoperability, application developers eventually came to package their DLLs in the form of Component Object Model (COM) libraries. Although these libraries do provide the capability to reuse code, they also suffer from limitations that .NET attempts to overcome.

As we've mentioned earlier in this book, one of the major problems with COM DLLs is that COM requires you to register a DLL on a machine before you can use it. This means that with a COM component, you can have only one version of a component installed on a machine at a time. This is a problem because there's no guarantee that future versions of a given component will provide the same functionality as the version you deploy your application with. (This is one aspect of the problem known as "DLL Hell.")

NOTE

Components and assemblies aren't technically the same thing. However, for the purposes of this (and most) discussions of components in Microsoft.NET, they are pretty close to the same thing. So we'll refer to components and assemblies more or less interchangeably in this section.

Microsoft.NET attempts to get around DLL Hell problems by encouraging developers to deploy dependent components in a subdirectory under their application directories. For ASP.NET applications, this means that components will be installed in a subdirectory called \bin under the application directory. But a problem occurs here. Component code that is accessible based on its physical location in the file system can't be shared across multiple applications. You could deploy multiple copies of the component to each application that required it (in fact, in the .NET world this may be preferable for compatibility reasons), but you may find yourself in a situation where you develop a tool or library whose functionality is so generically useful that you may want to provide access to that library in a global fashion.

Microsoft.NET provides a way do to this, called the Global Assembly Cache (GAC). Components that are installed into the GAC are available from any .NET application running on that computer. (Note that this includes other types of .NET applications, including Windows Forms applications, not just ASP.NET applications.)

This document is created with the unregistered version of CHM2PDF Pilot

You can install a component into the GAC using a command-line tool called gacutil. To install a component into the Global Assembly Cache, use the command

```
gacutil -i mycomp.dll
```

in which mycomp.dll is the name of your component. Note that this command will only work if the assembly has been signed; you sign an assembly using the sn.exe utility.

NOTE

You can create .NET components in Visual Studio. To do this, create a project using the "Class Library" project type.

To uninstall a component from the GAC, use the command

```
gacutil /u mycomp.dll
```

Be careful when using this command, because applications that rely on the component will break when it is removed from the GAC.

To see a list of the components in the GAC, use the command

```
gacutil /l
```

You'll see a list of all the components that have been installed on the system. Initially, this list will just be composed of Microsoft.NET assemblies (which you typically don't want to mess with).

Numerous configuration files and attributes can be configured quite easily and in a prompt manner. The Web.Config and Machine.Config files are the two most commonly used configuration files, and any changes to these files can be easily distributed by using the XCOPY functionality.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 6. Web Services

IN THIS CHAPTER

- Historical Influences
- Network Data Representation
- What Is a Web Service?
- Why Web Services?
- ASP.NET Web Services
- Consuming Web Services
- Class Reference

There is no way you could have existed on the planet earth in the year 2001 and not heard about the latest solution to all of mankind's problems—XML Web services. Depending on who you listen to, XML Web services will enable frictionless e-commerce or eliminate world hunger. So what are XML Web services, really? This chapter will tackle that question, plus look at why and how you should use XML Web services.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Historical Influences

Let's take a look back at the genesis of the Web. How did it start? It was the combination of a document format, HTML, and a protocol, HTTP, that enabled scientists to share documents in a standard fashion and to link those documents together. This was nothing new. We had a myriad of document formats: WordPerfect, Word, or even LATEX. The problem was that none of these document formats was interoperable. The guy who used WordPerfect couldn't read the LATEX documents, and vice versa. An interoperable document format alone, however, wouldn't solve the problem. A way also had to exist for scientists to discover papers published by other colleagues. This was done initially by placing hyperlinks into the document to enable navigation to other documents. Theoretically, given a starting point, a scientist could read a document and (by utilizing the hyperlinks) navigate to information related to the subject at hand. This navigation scheme assumed that there was a known starting point. This problem gave rise to the directory, such as Yahoo!, as a starting point. It was up to the directory to catalog the Web and indicate the appropriate starting point.

This clearly was a successful paradigm for finding, navigating to, and reading information. As the Internet grew, it became clear that a need existed for businesses to exchange information and transact business online. Although the Web was successful for humans to exchange information, it had far too little organization to make it an effective way for very literal-minded computers to take advantage of it.

What was appropriate (if not ideal) for a human being was far from ideal for a computer. First, computers need a fairly rigid structure to be applied to the information that they are exchanging. This structure must go beyond the document format to also encompass the structure and organization of the actual information itself. Second, if computers are going to trade information, there needs to be a way to agree on the format of information that is being exchanged. Finally, a need still exists to find partners to trade with. Given a partner, a business can negotiate with them to determine what services they may expose, but how does it find new partners? It still has a need for a directory service, but in this case it's one that the computer can query to find appropriate partners.

One answer to this problem is the concept of a Web service. This is in contrast to the ubiquitous Web page that we all know and love. A Web service is just what it sounds like: a facility that provides a way to do work. That being said, a Web service is not a Web page. It is not intended to be consumed by human eyes; it is intended to be consumed by a computer and is optimized for this type of access. If you want to make an analogy to the existing computer world, you could think of a Web service as a new form of Remote Procedure Call (RPC). Historically, the problem with RPC has been the lack of an open and widely accepted standard that defines how to represent data on the network, how to identify endpoint locations, and how to advertise the endpoints. Wait! This sounds very much like the problems just mentioned that the Web was created to solve! So let's take each of those three problems and think about how to translate the lessons of the Web to services.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Network Data Representation

One of the two big breakthroughs that enabled the Web was HTML. HTML was an open and standards-based data-formatting language that could be used to represent the data in a document. It was not a binary format but a text-based format based on the concept of markup "tags" that were inserted into the content to provide formatting. This had been done before. The Word file format is a binary form of formatting that holds both the content and the information required to format it. It, however, is not open nor standards based. Microsoft created it and controls it.

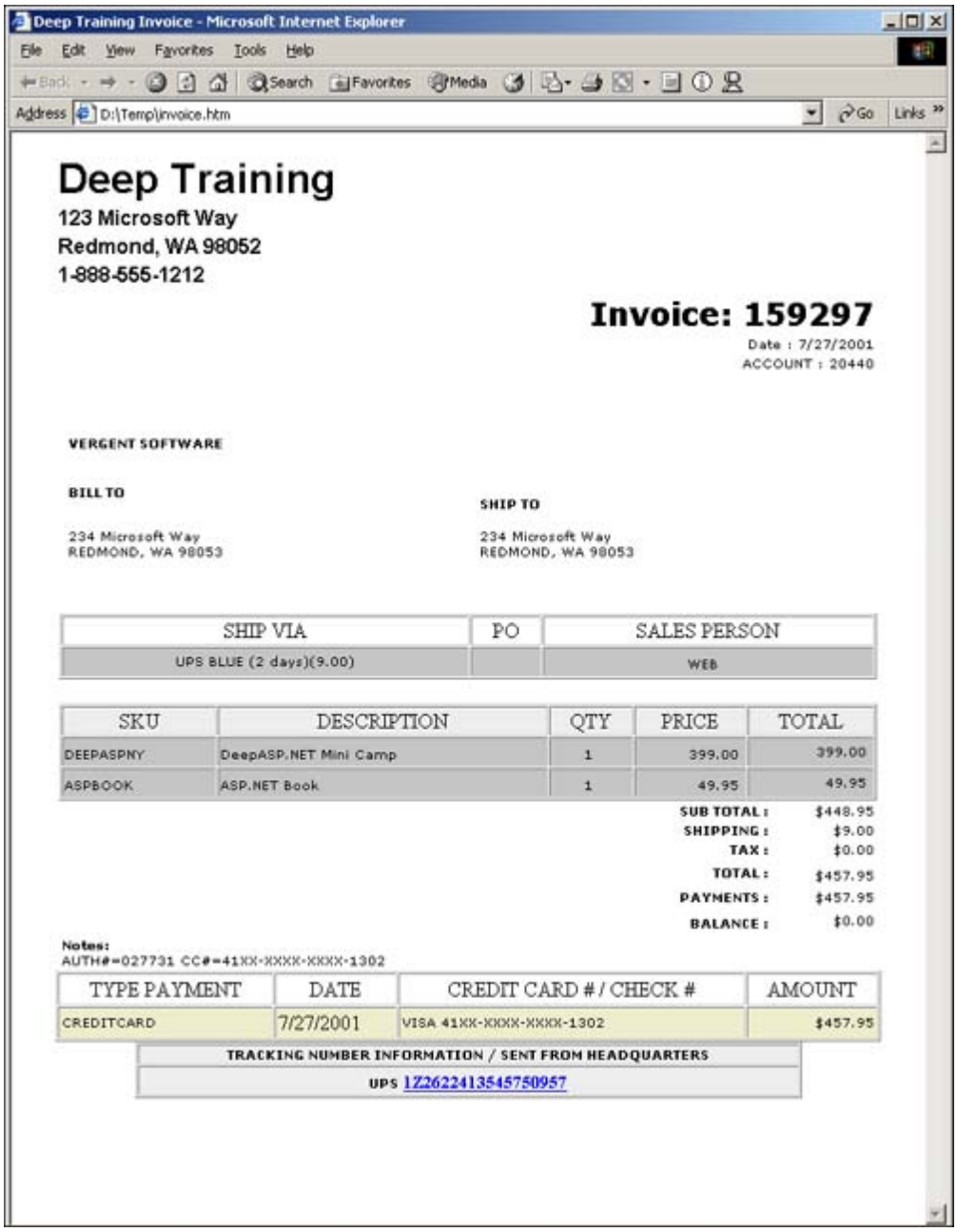
Perhaps more important is that it is binary. The barrier to entry for a binary format is that the user typically must create a program just to read or write the format. But with a text-based format such as HTML, anything that can create an ASCII text file can create and/or read the source of the format. The agreed upon format is to use ASCII or Unicode, which is a common standard, and to build on that by including inline markup tags.

How can this extend to the services model? HTML isn't a good fit because its primary mission is to control the formatting of content. Machines rarely care that a particular word is displayed in pink or blue. They are more concerned that the word itself is "pink" and what that might mean in a certain context. The idea, however, of using ASCII as a standard representation and then adding markup to create structure is a concept that can be generalized—and indeed has been—for something called eXtensible Markup Language (XML). XML is about the meaning of the document's content, as opposed to how the content is displayed.

Let's take a look at an example. I am going to express the same thing, an invoice, two ways. First off, let's look at a screenshot of the invoice. Figure 6.1 shows what the invoice would look like in the browser.

Figure 6.1. An invoice page in Internet Explorer.

This document is created with the unregistered version of CHM2PDF Pilot



This is what I would see as a human being browsing this Web page. What would I see if I were a computer browsing this page? I would see the underlying HTML markup. The same page in this format is shown in Listing 6.1.

Listing 6.1 The Same Invoice from Figure 6.1 Seen from a Computer's Perspective

```
<html>
<head>
<title>Deep Training Invoice</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<table cellSpacing="0" cellPadding="0" width="640" border="0" align="center">
  <tr>
    <td ALIGN="RIGHT">
      <table cellSpacing="1" cellPadding="1" width=640 border="0"
           align="center">
        <tr>
          <td><font face=Arial size=6><b>Deep Training</b></font></td>
        </tr>
        <tr>
          <td><font face=Arial size=3><b>123 Microsoft Way</b></font></td>
        </tr>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        <tr>
          <td><font face=Arial size=3><b>Redmond, WA 98052</b></font></td>
        </tr>
        <tr>
          <td><font face=Arial size=3><b>1-888-555-1212</b></font></td>
        </tr>
        <tr>
          <td></td>
        </tr>
      </table>
    </td>
  </tr>
  <tr>
    <td ALIGN="center">
      <table cellSpacing="1" cellPadding="1" width="99%" border="0"
           align="center">
        <tr>
          <td ALIGN="RIGHT"> <font face="Verdana" size="5">
               <b>Invoice: 159297</b></font>
          </td>
        </tr>
        <tr>
          <td align="right">
               <font face="Verdana" size="1">Date : 7/27/2001 </font>
          </td>
        </tr>
        <tr>
          <td align="right">
               <font face="Verdana" size="1">ACCOUNT : 20440 </font>
          </td>
        </tr>
      </table>
      <table cellSpacing="1" cellPadding="1" width="99%" border="0"
           align="center">
        <tr>
          <td> <BR>
          </td>
        </tr>
        <tr>
          <td>
            <table cellSpacing="1" cellPadding="1" width="99%" border="0"
                 align="center">
              <tr>
                <td align="left"> <font face="Verdana" size="-2">
                     <b>VERGENT SOFTWARE  
                  </b></font></td>
                <td>
                    <BR><BR>
<BR>
                </td>
              </tr>
              <tr>
                <td align="left"> <font face="Verdana" size="-2"> <b> BILL TO
                  </b></font><BR>
                  <BR>
                </td>
                <td>
                     <font face="Verdana" size="-2"> <b> SHIP TO </b></font>
                      <BR>
                </td>
              </tr>
              <tr>
                <td><font face="Verdana" size="-2">234 Microsoft Way<BR>
                  REDMOND, WA 98053 <br>
                  </font></td>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            <td><font face="Verdana" size="-2">234 Microsoft Way<BR>
              REDMOND, WA 98053 <br>
              </font></td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
  <BR>
  <BR>
  <table cellSpacing="1" cellPadding="1" width="99%" border="1"
       align="center">
    <tr align="center">
      <td>SHIP VIA</td>
      <td>PO</td>
      <td>SALES PERSON</td>
    </tr>
    <tr align="center" BGCOLOR="#c5c5c5">
      <td>
          <font face="Verdana" size="-2"> UPS BLUE (2 days)(23.51) </font>
      </td>
      <td><font face="Verdana" size="-2">   </font></td>
      <td><font face="Verdana" size="-2"> WEB  </font> </td>
    </tr>
  </table>
  <BR>
  <table cellSpacing="1" cellPadding="1" width="99%" align="center"
       border="1" bgcolor="#eeeeee">
    <tr>
      <td align="center">COURSE</td>
      <td align="center">DESCRIPTION</td>
      <td align="center">QTY</td>
      <td align="center">PRICE</td>
      <td align="center">TOTAL</td>
    </tr>
    <tr BGCOLOR="#c5c5c5">
      <td align="left">
           <font face="Verdana" size="-2">DEEPASPNY</font>
     </td>
      <td align="left">
           <font face="Verdana" size="-2">DeepASP.NET Mini Camp</font>
      </td>
      <td align="middle">
           <font face="Verdana" size="-2">1 </font>
      </td>
      <td align="right">
           <font face="Verdana" size="-2">399.00 </font>
      </td>
      <td align="right">
           <font face="Verdana" size="-2">399.00  </font>
      </td>
    </tr>
  </table>
  <table cellSpacing="1" cellPadding="1" width="99%" align="center"
        border="0">
    <tr>
      <td align="LEFT"> </td>
      <td align="right"> <font face="Verdana" size="-2"><b>SUB TOTAL :</b>
        </font> </td>
      <td align="right">
           <font face="Verdana" size="-2">$399.00 </font>
      </td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
      <td align="right">
           <font face="Verdana" size="-2"><b>(Non Taxable)OTHER
           CHARGES :</b> </font>
      </td>
      <td align="right"> <font face="Verdana" size="-2">$0.00 </font></td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
      <td align="right"> <font face="Verdana" size="-2"><b>DISCOUNT :</b>
        </font> </td>
      <td align="right"> <font face="Verdana" size="-2">$0.00</font> </td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
      <td align="right"> <font face="Verdana" size="-2">
           <b>FREIGHT :</b> </font>
      </td>
      <td align="right"> <font face="Verdana" size="-2">$0.00 </font></td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
      <td align="right">
           <font face="Verdana" size="-2"><b>TAX :</b> </font>
      </td>
      <td align="right"> <font face="Verdana" size="-2">$0.00 </font></td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
      <td align="right">
           <font face="Verdana" size="-2"><b>TOTAL :</b> </font>
      </td>
      <td align="right">
           <font face="Verdana" size="-2">$399.00</font>
      </td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
      <td align="right"> <font face="Verdana" size="-2"><b>PAYMENTS :</b>
        </font> </td>
      <td align="right">
           <font face="Verdana" size="-2">$399.00 </font>
      </td>
    </tr>
    <tr>
      <td align="LEFT"> </td>
      <td align="right">
           <font face="Verdana" size="-2"><b>BALANCE :</b></font>
      </td>
      <td align="right">
           <font face="Verdana" size="-2">$0.00 </font>
      </td>
    </tr>
    <tr>
      <td align="LEFT">
           <font face="Verdana" size="-2"><B>Notes:</B><BR>
           AUTH#=027731 CC#=41XX-XXXX-XXXX-1302 </font>
      </td>
    </tr>
  </table>
  <TABLE cellSpacing=1 cellPadding=2 border=1 width="80%"
       align="left">
    <TR>
      <TD ALIGN=MIDDLE>TYPE PAYMENT</TD>
      <TD align=middle>DATE</TD>
      <TD ALIGN=MIDDLE>CREDIT CARD # / CHECK #</TD>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
          <TD ALIGN=MIDDLE>AMOUNT</TD>
        </TR>
        <tr bgcolor="#eeeecc">
          <td><font face="Verdana" size="-2"> CREDITCARD </font></td>
          <td> 7/27/2001 </td>
          <td>
               <font face="Verdana" size="-2"> VISA 41XX-XXXX-XXXX-1302</font>
          </td>
          <td align=right><font face="Verdana" size="-2"> $399.00 </Font></td>
        </tr>
      </TABLE>
         </td>
         </tr>
</TABLE>
<TABLE cellSpacing=1 cellPadding=1 width="75%" align=center border=1
     bgcolor="#eeeeee">
  <TR>
    <TD ALIGN="CENTER">
         <font face="Verdana" size="-2"><B>TRACKING NUMBER INFORMATION
         / SENT FROM HEADQUARTERS</B></font>
    </TD>
  </TR>
  <TR>
    <TD ALIGN="CENTER"> <font face="Verdana" size="-2"> <b>UPS</b> </font><A
HREF=http://wwwapps.ups.com/tracking/tracking.cgi?tracknum=1Z2622413545750957
       target=new>
       <B><FONT size=2>1Z2622413545750957</FONT></b></A>
    </TD>
  </TR>
</TABLE>
</body>
</html>
```

Look at this HTML. Without the visual formatting, it is no longer nearly as easy to pick out the various pieces. How would you find the total or the authorization code? From a machine's perspective, this is mainly gobbledygook. I could say that the total is always going to come after a text string "TOTAL : </td><td align="right"> <font face="Verdana" size="-2">". But what happens when the developer of the page decides that the total should be shown in Helvetica? The string I am matching no longer works and my code breaks.

How can this be extended to a services model? To create a system whereby computers communicate without human intervention, HTML isn't going to cut it. It requires something that is more concerned with representing the data in a meaningful manner instead of making it look pretty. This is where XML comes in. Let's look at a representation of the same invoice in XML. Listing 6.2 shows one way to do it. XML is explained more thoroughly in Chapter 10, "Using XML."

Listing 6.2 A Representation of the Invoice in Listing 6.1 in XML

```
<?xml version="1.0" encoding="utf-8" ?>
<invoice number="159297" date="7272001">
    <account>20440</account>
    <company>Vergent Software</company>
    <billto>
        <address>234 Microsoft Way</address>
        <city>Redmond</city>
        <state>WA</state>
        <zip>98053</zip>
    </billto>
    <shipto>
        <address>234 Microsoft Way</address>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        <city>Redmond</city>
        <state>WA</state>
        <zip>98053</zip>
    </shipto>
    <shipvia>
        <transport>UPS Blue</transport>
        <days>2</days>
        <cost>9.00</cost>
        <tracking>1Z26224135457509 57</tracking>
    </shipvia>
    <salesperson>web</salesperson>
    <items>
        <item sku="DEEPASPNY">
            <description>DeepASP.NET Mini Camp</description>
            <qty>1</qty>
            <price>399.00</price>
        </item>
        <item sku="ASPBOOK">
            <description>ASP.NET Book</description>
            <qty>1</qty>
            <price>49.95</price>
        </item>
    </items>
    <subtotal>448.95</subtotal>
    <shipping>9.00</shipping>
    <tax>0.00</tax>
    <total>457.95</total>
    <payments>457.95</payments>
    <balance>0.00</balance>
    <paymenttype>CREDITCARD</paymenttype>
    <creditcard>
        <type>VISA</type>
        <number>43XX-XXXX-XXXX-1302</number>
        <auth>027731</auth>
        <date>07-27-2001</date>
        <amount>457.95</amount>
    </creditcard>
</invoice>
```

Now is it clear where the total for this invoice is? It is enclosed by the <total> and </total> tags. These are tags totally unrelated to the display of the information. Their only purpose is to define where to look in the document to find the total. This makes them great candidates for string matching to pick apart the document in a standard way.

Location

How do I define the location or endpoint of a page on the World Wide Web? The Web popularized the concept of a URL, or uniform resource locator. You have seen these. They are strings such as http://www.deeptraining.com/default.aspx. The URL in the preceding example is made up of several parts. A syntax-style definition of a URL is as follows:

```
<protocol> "://" <host> [":" <port>] [<path> ["?" <query>]]
```

The first part identifies the protocol. The HTTP at the beginning of the earlier example means that when accessing this URL, you should use the Hypertext Transfer Protocol. Another valid protocol identifier for most browsers is FTP, or File Transfer Protocol. Internet Explorer accepts either

```
file://c:\temp\invoice.htm
```

This document is created with the unregistered version of CHM2PDF Pilot

or

```
ftp://localhost/temp/invoice.htm
```

The second part identifies the host that contains the resource. This is permitted to contain an IP address, but in most cases, it will contain a hostname.domain.network combo such as www.deeptraining.com. The third part is an optional port designation. If not specified, the default convention is to use port 80 for all HTTP traffic. By specifying a port, you can potentially host more than one Web server on a single IP address. This is frequently used by network address translation (NAT)-based firewalls to direct incoming traffic to Web servers behind the firewall. The fourth part is one of the more important parts. It indicates the path to the resource. This is a standard path of the form /temp/invoice.htm. Note the forward slashes used in the path. The HTTP protocol was invented in the Unix world in which path delimiters are forward-slash characters, in contrast to the backslash characters used in the DOS/Windows world. The last part is optional information that varies for a particular path. You have seen this when you go to a search page. You type in what you are interested in and a page is displayed with a URL like

http://www.deeptraining.com/searchresults.aspx?Query=ASP.

The ?Query=ASP part on the end is a query parameter used to pass additional information to the search results page.

The combination of all these parts represents a unique endpoint in the scheme of the universe. In addition, it is an endpoint that even my 8-year-old daughter can attribute some meaning to, given the ubiquity of Web usage in today's Internet-savvy world.

In a world where I want to make services available, URLs are useful to uniquely identify the location of my service. I can also potentially use the query parameters portion of the URL to optionally pass information to my service.

Advertisement

How do you find information on the wildly popular Ichiro Suzuki bobblehead doll? If you are like most people today, you fire up a Web browser and look it up. But how do you find the information? Your first try is probably to go to www.ichirosuzuki.com or perhaps even www.seattlemariners.com. If that didn't have the information you were looking for, what is the next step? You can head to a search engine such as www.google.com and type in "Ichiro Bobblehead." In no time at all, Google will spit back dozens of matches for Web sites that have information on the latest craze to hit Safeco field.

Let's translate this to the idea of services. I have a great Web site that I built recently to sell some of those Ichiro bobblehead dolls. When billing the customers an exorbitant amount, I want to make sure that I also charge a sufficient amount for shipping. It would make sense that given the shipping address I need to send the doll to, I want to calculate how much it is going to cost to ship it. I want to utilize a Web service to do this in real-time. I know I am going to be shipping the dolls to eagerly waiting customers using United Parcel Services (UPS) and need to find a service that calculates UPS 2-day rates.

This document is created with the unregistered version of CHM2PDF Pilot

My first guess is to go to www.ups.com, but I quickly determine that they don't yet offer UPS ground rate calculation as a Web service. How can I find out who else might? This is where a search engine analogous to Google would be valuable. As it turns out, several vendors are building directories of services that allow a developer to query them and discover trading partners that offer the services they are interested in. These directories provide a standard interface—Universal Description, Discovery, and Integration (UDDI)—for the categorization of services, companies, and the schemas they use. They are accessible via a Web-based interface for you to initially find the services that will fulfill your needs. The UDDI directories also expose themselves using XML Web services so that your applications can dynamically use them also.

After I have a reference to a server, I also need to be able to determine what services that particular server exposes to the outside world. This browsing of services is facilitated by placing an XML file, called a DISCO file, in the root of the Web server. DISCO stands for Discovery, and this XML file provides links to all the XML Web services exposed on that server.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

What Is a Web Service?

So what is a Web service? You probably have some ideas based on the parallels I have drawn with the genesis of the World Wide Web. XML Web services are an open way to perform standards-based remote-procedure calls over standard Internet protocols. Wow—that's quite a mouthful. I am sure more than one person will argue with that definition because it is difficult to encapsulate the idea of something as large as XML Web services into a single sentence.

So let's examine more closely what a Web service is. XML Web services are not Web pages. They are intended to be created and consumed by applications, not users. Instead of designing the look, you define the schema. The schema is what is important in a Web service.

Several standards apply to XML Web services today. XML Web services communicate over HTTP/1.1, a standard protocol defined in RFC 2616 and RFC 2068. The data that is passed back and forth between XML Web services is encapsulated in XML, which is a W3C recommendation at the present time. Simple Object Access Protocol (SOAP) is an XML grammar that defines the layout of the requests sent to XML Web services and the response received back. As of this writing, SOAP is at W3C NOTE status and more information can be found at http://www.w3.org/TR/SOAP/. Web Services Description Language (WSDL) is another XML grammar for defining the application-specific content of SOAP requests and responses. Universal Description, Discovery, and Integration (UDDI) is a standard protocol for quickly and easily finding XML Web services run by trading and business partners.

Whew! Those are quite a few standards. I also said that XML Web services were open. At their very simplest, XML Web services are a way to take SOAP (XML text) and send it over HTTP. This means that any language that is capable of performing TCP socket operations and string manipulation can play in this space. Granted, if TCP sockets and string manipulation were all you had at your fingertips, it would be like deciding to dig a swimming pool in your backyard with a teaspoon. It would not be trivial, but it would definitely be possible. Fortunately, a number of development platforms, including .NET, are building a significant amount of the infrastructure to make creating and consuming XML Web services trivial.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Why Web Services?

The concept of RPC is nothing new. In the preceding years we have been proselytized to use DCOM, CORBA, and a number of other RPC protocols. However, none of these protocols has received enough support to make them ubiquitous and thus enable any trading between partners. DCOM and CORBA both used their own data representations that, while similar in many respects, are different enough to prevent any interoperation. They each define their own protocols that don't work very well in a high-latency WAN such as the Internet. DCOM in particular is very "chatty," requiring numerous round trips just to negotiate a simple remote procedure call with no arguments. In addition, with the paranoid mentality of corporations connecting to the Internet through firewalls and the like, the chances are slim of either an IIOP or DCOM request making it through a firewall. Finally, DCOM is a connection-oriented protocol. The downside of this is that after a connection has been set up, DCOM expects to have a long-running conversation with the remote object, making load balancing and load farming a difficult proposition at best.

On the other hand, the underlying protocol of XML Web services, HTTP, has had untold millions of dollars spent on it in the last few years to solve the problems of scalability and fault tolerance in support of the boom in the Web space during the mid 1990s. Well-known best practices exist for scaling HTTP by creating farms of Web servers, using dynamic location-based DNS, and even performing switching in layers 2–7 of TCP/IP to support quite intelligent load balancing. All this work can now be applied to XML Web services.

With the creation of SOAP came a standard for an XML grammar that can be used to overcome the differences that plagued the various RPC implementations in the past. SOAP defines how any data type, for example, an int, should be encoded, regardless of platform.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

ASP.NET Web Services

All this is great, but as a Web developer, I don't want to have to go out and learn HTTP, XML, SOAP, WSDL, and DISCO just so that I can trade with my partners. I don't have the time. ASP.NET to the rescue.

The model for creating XML Web services in ASP.NET is very similar to the model for creating programmable pages. Let's create a very simple Web service and look at what it is composed of. In its simplest form, a Web service is a file with an extension, ASMX, that is new to ASP.NET. As you would expect, no HTML is in this page, only code. Listing 6.3 shows the canonical HelloWorld that in some shape or another tends to be every programmer's first application.

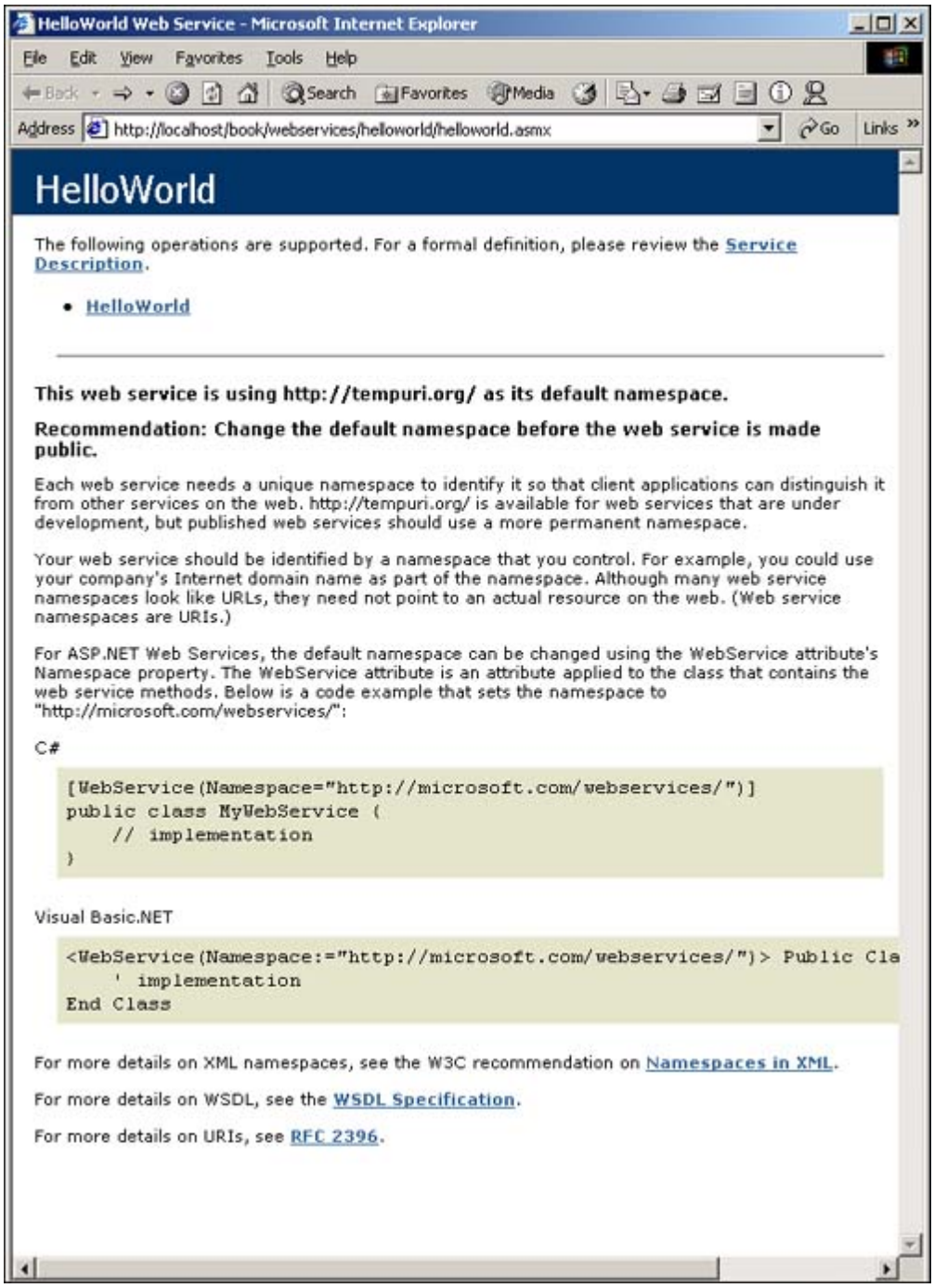
Listing 6.3 A Simple Web Service Saying Hello to the World

```
<%@ WebService Language="c#" Class="HelloWorldSvc" %>

using System.Web.Services;

public class HelloWorldSvc : System.Web.Services.WebService
{
    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World!";
    }
}
```

That's it! After all the talk of SOAP, XML, and so on, this looks just like a standard class. The .NET framework hides the ugly part of creating XML Web services from you, the developer, allowing you to concentrate on what you need the Web service to do instead of how it does it. Well, the code is cool, but we want to see it do something. Remember that what we just wrote is intended to be called by a program, not by a user. Having to immediately write a test harness just to test a simple Web service is kind of a pain. Consequently, the .NET framework provides a default test harness that will appear if you enter the URL for a Web service endpoint into the browser. If a particular method in a Web service is not specified, it is assumed that the end user needs some more information about the Web service. If I enter http://localhost/book/webservices/helloworld/HelloWorld.asmx, the address for the Web service in Listing 6.3, I get the browser display shown in Figure 6.2.

Figure 6.2. The automatically created documentation page.

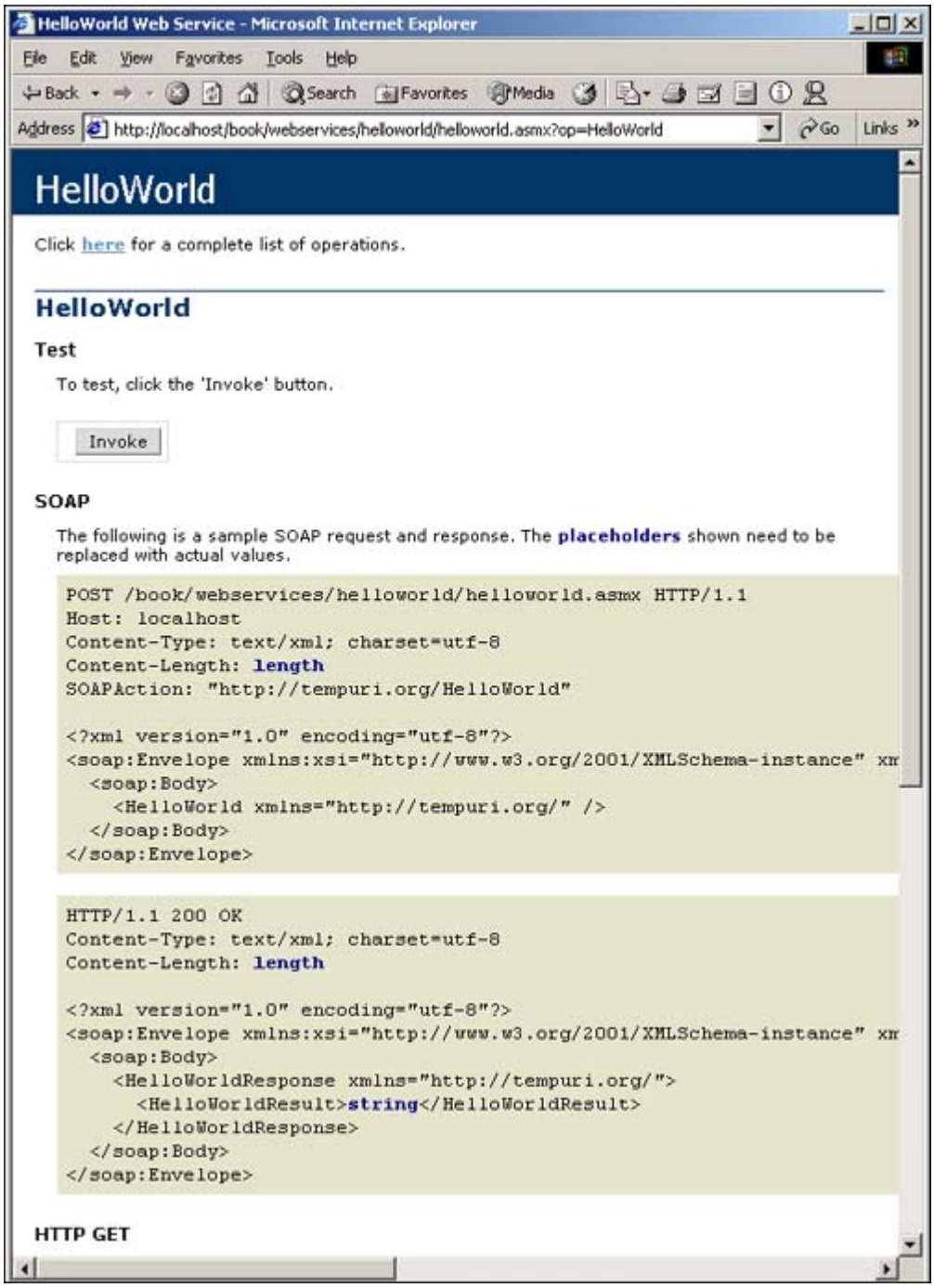
This document is created with the unregistered version of CHM2PDF Pilot


HelloWorld Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/book/webservices/helloworld/helloworld.asmx
Go Links
HelloWorld
The following operations are supported. For a formal definition, please review the Service Description.
HelloWorld
This web service is using http://tempuri.org/ as its default namespace.
Recommendation: Change the default namespace before the web service is made public.
Each web service needs a unique namespace to identify it so that client applications can distinguish it from other services on the web. http://tempuri.org/ is available for web services that are under development, but published web services should use a more permanent namespace.
Your web service should be identified by a namespace that you control. For example, you could use your company's Internet domain name as part of the namespace. Although many web service namespaces look like URLs, they need not point to an actual resource on the web. (Web service namespaces are URIs.)
For ASP.NET Web Services, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":
C#
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
// implementation
}
Visual Basic.NET
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Cla
' implementation
End Class
For more details on XML namespaces, see the W3C recommendation on Namespaces in XML.
For more details on WSDL, see the WSDL Specification.
For more details on URIs, see RFC 2396.

This page gives some general information about the Web service, including the methods in my Web service. If I click the method name, I get the page shown in Figure 6.3.

Figure 6.3. The automatically created test harness.

This document is created with the unregistered version of CHM2PDF Pilot



This page gives me a way to invoke the method, and it documents the appropriate ways to call my method using SOAP, HTTP GET, and HTTP POST. If I click the Invoke button, my Web service is called using HTTP GET, and I receive a response back that is shown in Figure 6.4.

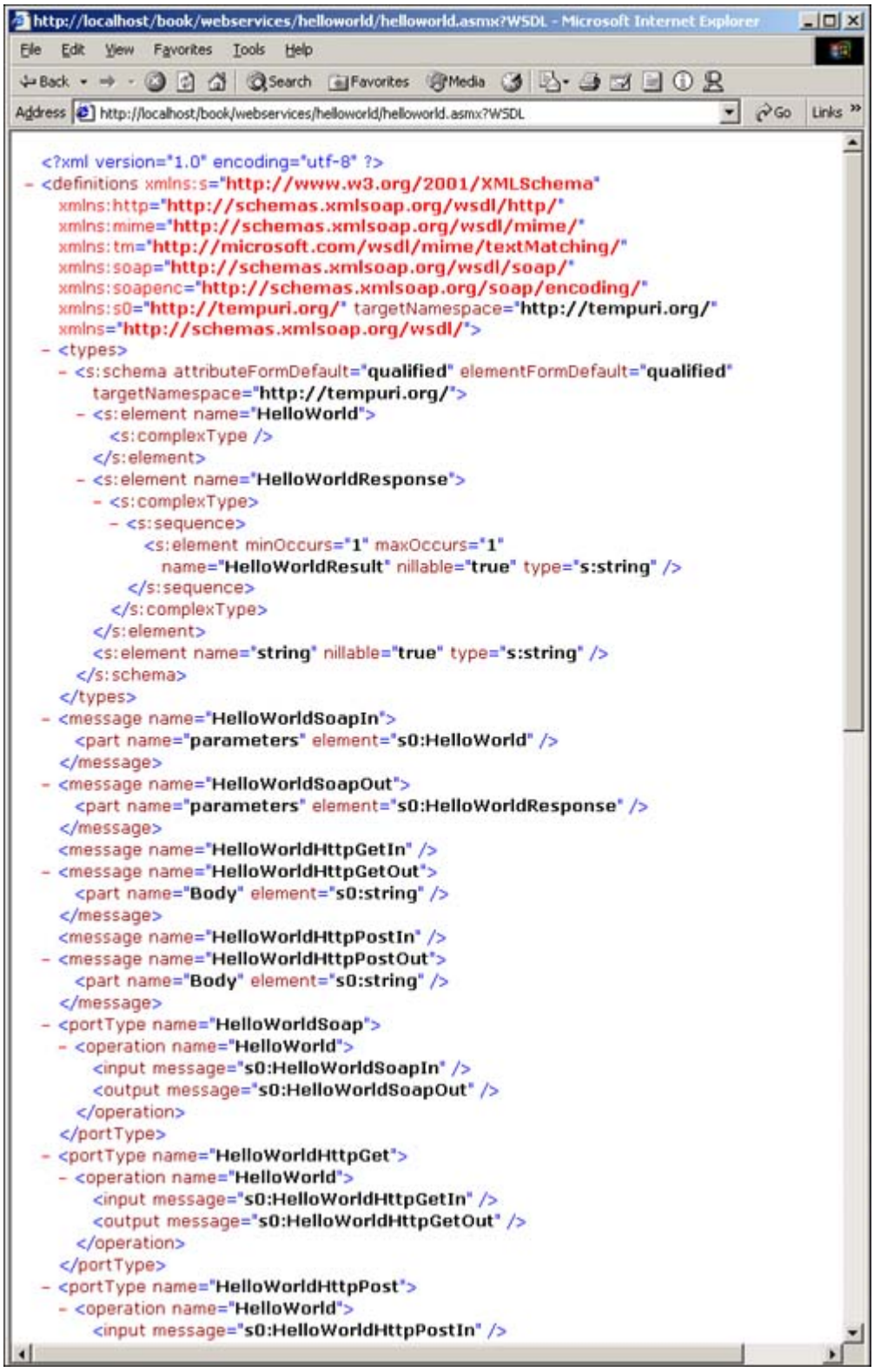
Figure 6.4. The XML returned by calling the Web method via HTTP GET.

This document is created with the unregistered version of CHM2PDF Pilot

One thing shown in Listing 6.3 does look a little unusual, though. The keyword that looks like [WebMethod] is called an attribute in .NET. Attributes are a way for the programmer to declaratively affect the operation of an application without having to write code. This particular attribute does a considerable amount of work. The WebMethod() attribute is somewhat similar to the standard access modifier public. By placing the WebMethod() attribute on my method, I have indicated that I want to make it publicly callable from the outside world. Only functions with WebMethod() are accessible by clients of the Web service. This restriction allows me to continue to have internal methods that I rely on within the class, without having to worry about them being accidentally called by clients. Specifying WebMethod() also tells .NET that it should include this method in the WSDL that it generates for clients. WSDL is the way that clients are going to figure out the proper way to call my methods. To see the WSDL that is automatically generated by .NET, I can call my Web service (shown in Listing 6.3) with this URL: http://localhost/book/webservices/helloworld/helloworld.asmx?WSDL. The output of this is shown in Figure 6.5.

Figure 6.5. The Web Services Description Language (WSDL) output from the Web service.

This document is created with the unregistered version of CHM2PDF Pilot



NOTE

This URL points to where I placed the sample code on my system and may vary on your system, depending on where you save the source code.

You can see the HelloWorld method clearly delineated in the WSDL. We will take a look at what the WSDL is good for when we show how to consume XML Web services. The WebMethod attribute also provides a way to configure several optional attributes on a Web service.

This document is created with the unregistered version of CHM2PDF Pilot

WebMethodAttribute

The WebMethodAttribute class is what we are using with the WebMethod() attribute mentioned previously. WebMethodAttribute is used to set the options for a Web method.

The BufferResponse property controls how data is sent back to the client from the Web service. The most efficient method for returning data over TCP is to batch it all up and send it in large blocks to the client. This is what is considered buffered mode and is the default both for Web pages and XML Web services. In the case of a large database query, however, it might be nice to start streaming the contents back to the client before the query has finished retrieving all the rows. To do this, set buffer response to false. When buffering is turned off, the results are sent back to the client in 16KB chunks.

The EnableSession property enables session state for a Web service. By default, this attribute is set to false. Think hard about whether you need to enable session state on a Web service, because storing state on the server is going to affect the scalability of your service. However, session state can be utilized with all the attendant options, as discussed in Chapter 4, "State Management and Caching." This includes the State Server and SQL Server modes that are new to ASP.NET.

Listings 6.4 and 6.5 include a sample Web service that provides a Web service–based state service. The service provides two methods, SetValue and GetValue. SetValue enables the user to store some data with a keyname and a value. GetValue enables the user to retrieve data with a keyname. This example also uses the code-behind model (discussed in Chapter 2, "Page Framework") for creating XML Web services. As you can see, the activator for the Web service, the .asmx file, is minimal. In all future examples, I won't even include the .asmx as part of the listing.

Listing 6.4 The Activator .asmx File

```
<%@ WebService Language="c#" Codebehind="State.asmx.cs" Class="HelloWorld.State" %>
```

Listing 6.5 The Code-Behind File for the Simple State Service

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace HelloWorld
{
    /// <summary>
    /// Summary description for State.
    /// </summary>
    public class State : System.Web.Services.WebService
    {
        public State()
        {
            //CODEGEN: This call is required by the ASP.NET Web Services Designer
            InitializeComponent();
        }

        #region Component Designer generated code
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        }
        #endregion

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
        }

        [WebMethod(true)]
        public void SetValue(string Name, string Value)
        {
            Session[Name] = Value;
        }

        [WebMethod(true)]
        public string GetValue(string Name)
        {
            return Session[Name].ToString();
        }
    }
}
```

The Description property supplies a description, which is shown in the Web service help page that is created as an automatic test harness. Listing 6.6 shows the code-behind class for the War Games Web service.

Listing 6.6 A Web Service the That Utilizes the Description Property

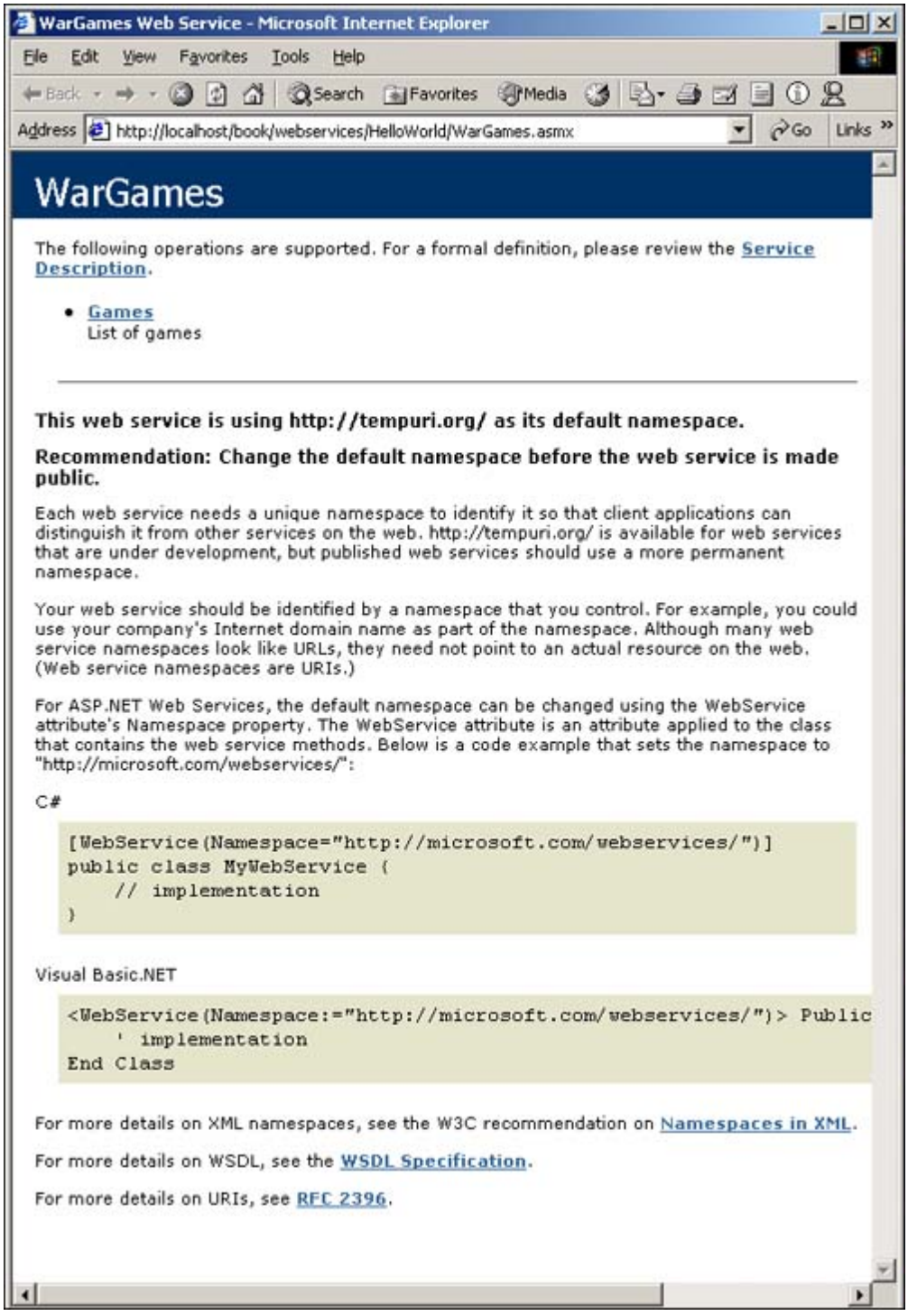
```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace HelloWorld
{
    public class WarGames : System.Web.Services.WebService
    {
        protected override void Dispose( bool disposing )
        {
        }

        [WebMethod(Description="List of games")]
        public string Games()
        {
            return "Tic Tac Toe, Chess, Thermonuclear War";
        }
    }
}
```

The WebMethodAttribute uses the Description property to indicate what each Web method does. The Description

This document is created with the unregistered version of CHM2PDF Pilot

property is set using the syntax for named properties in an attribute. Figure 6.6 shows how the Description property conveniently identifies the Games Web method so we know that it returns a list of games we can play with the WOPR.

Figure 6.6. The test harness page when we hack into War Games.


WarGames Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/book/webservices/HelloWorld/WarGames.asmx
Go Links
WarGames
The following operations are supported. For a formal definition, please review the Service Description.
Games
List of games
This web service is using http://tempuri.org/ as its default namespace.
Recommendation: Change the default namespace before the web service is made public.
Each web service needs a unique namespace to identify it so that client applications can distinguish it from other services on the web. http://tempuri.org/ is available for web services that are under development, but published web services should use a more permanent namespace.
Your web service should be identified by a namespace that you control. For example, you could use your company's Internet domain name as part of the namespace. Although many web service namespaces look like URLs, they need not point to an actual resource on the web. (Web service namespaces are URIs.)
For ASP.NET Web Services, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":
C#
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
// implementation
}
Visual Basic.NET
<WebService(Namespace:="http://microsoft.com/webservices/")> Public
' implementation
End Class
For more details on XML namespaces, see the W3C recommendation on Namespaces in XML.
For more details on WSDL, see the WSDL Specification.
For more details on URIs, see RFC 2396.

The CacheDuration property controls how a Web service is cached. The default for cache duration is 0, meaning that no caching is performed. As mentioned in Chapter 4, "State Management and Caching," huge performance increases can be realized by utilizing caching. The Cache[] object discussed in Chapter 4 is also available in XML Web services. The CacheDuration property is analogous to OutputCaching in a Web page. When this is set to some number of seconds, all output from the Web service is cached for this period of time. Listing 6.7 shows an example of a time service that only updates its output every 60 seconds.

Listing 6.7 The Code Behind Web Service Class That Implements a Cached Time Service

This document is created with the unregistered version of CHM2PDF Pilot

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace HelloWorld
{
    public class Time : System.Web.Services.WebService
    {
        protected override void Dispose( bool disposing )
        {
        }

        [WebMethod(CacheDuration=60)]
        public string GetTime()
        {
            return DateTime.Now.ToString();
        }
    }
}
```

NOTE

If this doesn't appear to work, make sure you are using the test harness outside Visual Studio .NET. Visual Studio .NET performs cache busting that penetrates the cache. The correct behavior will be displayed in Internet Explorer.

The TransactionOption property controls how the Web service interacts with the transaction-processing support found in the common language runtime. By altering the attribute of the Web method, you can control how the method participates in transactions. The default setting for the transaction option is Disabled. This means that by default, a Web method does not participate in any transactions. A Web service is limited to acting as the root in a transaction in version 1.0 of .NET. This limitation means that several of the transaction options provide the same functionality. Required and RequiresNew do the same thing because the Web method must be the root. This possibly could change in future versions. By default, the System.EnterpriseServices assembly is not referenced in VS.NET. To use the TransactionOption enumeration you will need to add a reference.

Serialization

All our samples so far have utilized relatively simple data types. As it turns out, almost any object in .NET can be serialized to XML. This includes Collections, Arrays, and even DataSets. One exception to this rule is any object that is based on System.Collections.Hashtable internally. This includes many of the dictionary-based collections in the base class library with the exception of ListDictionary. Serialization is the process whereby a running object provides a static representation of itself that can be used to later reconstitute this object and create a new running object.

Listing 6.8 shows a Web service that returns an ArrayList of shipping methods.

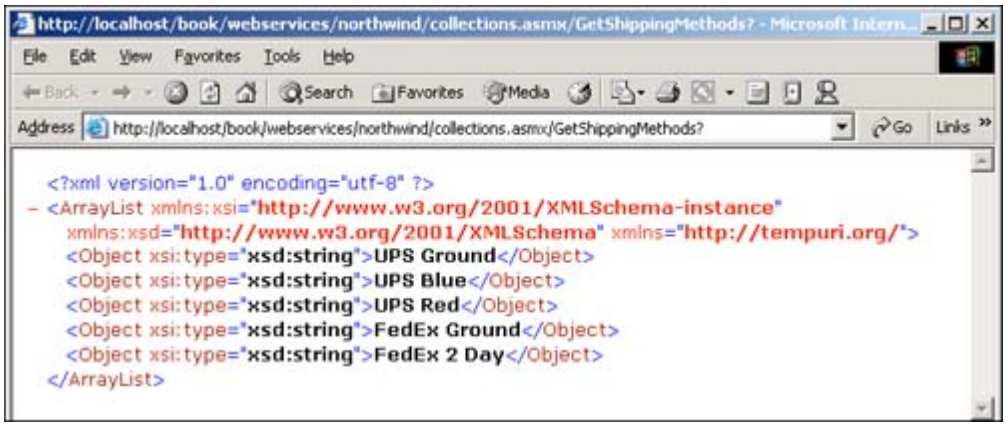
Listing 6.8 A Web Service That Returns an ArrayList

This document is created with the unregistered version of CHM2PDF Pilot

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace NorthwindOrders
{
    public class collections : System.Web.Services.WebService
    {
        protected override void Dispose( bool disposing )
        {
        }

        [WebMethod()]
        public ArrayList GetShippingMethods()
        {
            ArrayList al = new ArrayList();

            al.Add("UPS Ground");
            al.Add("UPS Blue");
            al.Add("UPS Red");
            al.Add("FedEx Ground");
            al.Add("FedEx 2 Day");

            return al;
        }
    }
}
```

Figure 6.7 shows what the returned XML looks like.

Figure 6.7. The output XML from the Web service that serializes the ArrayList.



The most interesting data type in my mind, however, is the DataSet. The DataSet is a new feature of ADO.NET that appears to be a perfect data structure for transporting data between XML Web services and client code. A DataSet has schema, which is just like a database. This schema defines the tables, their columns, and the relationship between tables within the DataSet. In this chapter, we aren't going to discuss all the features of DataSets. For more in-depth information on DataSets, see Chapter 11, "Creating Database Applications with ADO.NET."

We are going to look at the ways in which data sets can be used to move data between a Web service and a client.

This document is created with the unregistered version of CHM2PDF Pilot

Let's look at a simple case first. The SimpleDataSet example has a single WebMethod Simple() showing in Listing 6.9 . This method returns a list of orders in a DataSet. It first builds up the orders by creating a DataSet from scratch. It then creates a DataTable and adds columns to it. Each of these columns is strongly typed. The typeof() operator is used to get a type class to pass to the constructor for the DataColumn. After the DataTable has been created, we add rows to it. Calling NewRow() gives us a new row template whose strong typing is based on the DataTable it came from. Finally, the DataTable is added to the DataSet, and the DataSet is returned to the client. The complete WebMethod can be seen in Listing 6.9.

Listing 6.9 The Simple() WebMethod of SimpleDataSet. It Returns a DataSet Containing a Strongly Typed DataTable

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace SimpleDataSet
{
    public class DataSetSample : System.Web.Services.WebService
    {
        protected override void Dispose( bool disposing )
        {
        }

        [WebMethod()]
        public DataSet Simple()
        {
            System.Data.DataSet dsOrders = new System.Data.DataSet();
            DataTable dt;

            // Build a dataset with four columns
            dt = new DataTable("Orders");
            DataColumn dc = new DataColumn("OrderID", typeof(string));
            dt.Columns.Add(dc);
            dc = new DataColumn("Date", typeof(string));
            dt.Columns.Add(dc);
            dc = new DataColumn("Name", typeof(string));
            dt.Columns.Add(dc);
            dc = new DataColumn("Amount", typeof(decimal));
            dt.Columns.Add(dc);

            // Populate the dataset
            DataRow dr;
            dr = dt.NewRow();

            dr["OrderID"] = System.Guid.NewGuid();
            dr["Date"] = DateTime.Now;
            dr["Name"] = "Chris Kinsman";
            dr["Amount"] = 123.45;
            dt.Rows.Add(dr);

            dr = dt.NewRow();
            dr["OrderID"] = System.Guid.NewGuid();
            dr["Date"] = DateTime.Now.AddDays(1);
            dr["Name"] = "Jeffrey McManus";
            dr["Amount"] = "234.45";
            dt.Rows.Add(dr);

            // Add the datatable to the dataset
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            dsOrders.Tables.Add(dt);

            return dsOrders;

        }
    }
}
```

Figure 6.8 shows the output from this Web service. It starts with the Schema information for the dataset that we are returning. It defines each of the columns along with the data types. After this section, it uses the predefined schema to represent the data. You can pick out each or the Order rows along with each of the columns data quite easily. It should be quite evident that it would be simple to consume this data in a rigorous fashion.

Figure 6.8. The XML output from SimpleDataSet.

This document is created with the unregistered version of CHM2PDF Pilot

Let's do a little bit more complex example now. A common data construct is the idea of a Master-Detail relationship. You saw one of these when we were looking at the XML for the invoice. For an order (the master) I had multiple items (the detail). This type of relationship is common in databases, and any method of transferring data must take relationships into account. The example in Listing 6.10 will also return order data; however, this time we will utilize the Northwind database that ships with SQL Server as the source for our data. Listing 6.10 shows the new Web method.

Listing 6.10 A Web Method That Returns a Dataset with a Master Detail Relationship

```
[WebMethod()]
public DataSet GetOrders(DateTime OrderDate)
{
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    // Setup the connection
    SqlConnection cn = new SqlConnection(Application["DSN"].ToString());
    // Open the connection
    cn.Open();
    // Create the orders data adapter
    SqlDataAdapter daOrders = new SqlDataAdapter("SELECT * FROM ORDERS WHERE  OrderDate
= '" + OrderDate.ToShortDateString() + "'", cn);

    // Create the order item data adapter
    SqlDataAdapter daOrderDetails = new SqlDataAdapter("SELECT * FROM [Order Details]
od, Orders o WHERE o.OrderID = od.OrderID AND o.OrderDate = '" +
OrderDate.ToShortDateString() + "'", cn);

    // Create a data set
    DataSet ds = new DataSet();

    // Get the orders
    daOrders.Fill(ds, "Orders");
    // Get the order details
    daOrderDetails.Fill(ds, "OrderDetails");

    // Relate the two on the order id
    ds.Relations.Add("OrderID", ds.Tables["Orders"].Columns["OrderID"],
ds.Tables["OrderDetails"].Columns["OrderID"]);

    // Return the dataset
    return ds;
}
```

This code is somewhat similar to the previous example, but a few differences exist. First, we are using the ADO.NET SqlClient to retrieve the data (for more information on this, see Chapter 11). Second, we are not returning just a single table containing data. We are retrieving all the order and order details for all orders that were placed on OrderDate. The database defines a relationship between these two DataTables on the OrderID column that is present in each of the tables. This yields a DataSet that not only contains the data that matches the criteria from both tables, but also knows about the relationship between the data. Listing 6.11 shows the output when we call the GetOrders WebMethod witha date of a date of 7/8/1996.

Listing 6.11 The XML Output from Calling the GetOrders WebMethod with a Date of 7/8/1996

```
  <?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://tempuri.org/">
- <xsd:schema id="NewDataSet" targetNamespace="" xmlns=""
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
- <xsd:element name="NewDataSet" msdata:IsDataSet="true">
- <xsd:complexType>
- <xsd:choice maxOccurs="unbounded">
- <xsd:element name="Orders">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element name="OrderID" type="xsd:int" minOccurs="0" />
  <xsd:element name="CustomerID" type="xsd:string" minOccurs="0" />
  <xsd:element name="EmployeeID" type="xsd:int" minOccurs="0" />
  <xsd:element name="OrderDate" type="xsd:dateTime" minOccurs="0" />
  <xsd:element name="RequiredDate" type="xsd:dateTime" minOccurs="0" />
  <xsd:element name="ShippedDate" type="xsd:dateTime" minOccurs="0" />
  <xsd:element name="ShipVia" type="xsd:int" minOccurs="0" />
  <xsd:element name="Freight" type="xsd:decimal" minOccurs="0" />
  <xsd:element name="ShipName" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipAddress" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipCity" type="xsd:string" minOccurs="0" />
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  <xsd:element name="ShipRegion" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipPostalCode" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipCountry" type="xsd:string" minOccurs="0" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="OrderDetails">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element name="OrderID" type="xsd:int" minOccurs="0" />
  <xsd:element name="ProductID" type="xsd:int" minOccurs="0" />
  <xsd:element name="UnitPrice" type="xsd:decimal" minOccurs="0" />
  <xsd:element name="Quantity" type="xsd:short" minOccurs="0" />
  <xsd:element name="Discount" type="xsd:float" minOccurs="0" />
  <xsd:element name="OrderID1" type="xsd:int" minOccurs="0" />
  <xsd:element name="CustomerID" type="xsd:string" minOccurs="0" />
  <xsd:element name="EmployeeID" type="xsd:int" minOccurs="0" />
  <xsd:element name="OrderDate" type="xsd:dateTime" minOccurs="0" />
  <xsd:element name="RequiredDate" type="xsd:dateTime" minOccurs="0" />
  <xsd:element name="ShippedDate" type="xsd:dateTime" minOccurs="0" />
  <xsd:element name="ShipVia" type="xsd:int" minOccurs="0" />
  <xsd:element name="Freight" type="xsd:decimal" minOccurs="0" />
  <xsd:element name="ShipName" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipAddress" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipCity" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipRegion" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipPostalCode" type="xsd:string" minOccurs="0" />
  <xsd:element name="ShipCountry" type="xsd:string" minOccurs="0" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:choice>
  </xsd:complexType>
- <xsd:unique name="Constraint1">
  <xsd:selector xpath=".//Orders" />
  <xsd:field xpath="OrderID" />
  </xsd:unique>
- <xsd:keyref name="OrderID" refer="Constraint1">
  <xsd:selector xpath=".//OrderDetails" />
  <xsd:field xpath="OrderID" />
  </xsd:keyref>
  </xsd:element>
  </xsd:schema>
- <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <NewDataSet xmlns="">
- <Orders diffgr:id="Orders1" msdata:rowOrder="0">
  <OrderID>10250</OrderID>
  <CustomerID>HANAR</CustomerID>
  <EmployeeID>4</EmployeeID>
  <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
  <RequiredDate>1996-08-05T00:00:00.0000000-07:00</RequiredDate>
  <ShippedDate>1996-07-12T00:00:00.0000000-07:00</ShippedDate>
  <ShipVia>2</ShipVia>
  <Freight>65.83</Freight>
  <ShipName>Hanari Carnes</ShipName>
  <ShipAddress>Rua do Paço, 67</ShipAddress>
  <ShipCity>Rio de Janeiro</ShipCity>
  <ShipRegion>RJ</ShipRegion>
  <ShipPostalCode>05454-876</ShipPostalCode>
  <ShipCountry>Brazil</ShipCountry>
  </Orders>
- <Orders diffgr:id="Orders2" msdata:rowOrder="1">
  <OrderID>10251</OrderID>
  <CustomerID>VICTE</CustomerID>
```

This document is created with the unregistered version of CHM2PDF Pilot


```
    <EmployeeID>3</EmployeeID>
    <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
    <RequiredDate>1996-08-22T00:00:00.0000000-07:00</RequiredDate>
    <ShippedDate>1996-07-15T00:00:00.0000000-07:00</ShippedDate>
    <ShipVia>1</ShipVia>
    <Freight>41.5</Freight>
    <ShipName>Victuailles en stock</ShipName>
    <ShipAddress>2, rue du Commerce</ShipAddress>
    <ShipCity>Lyon</ShipCity>
    <ShipPostalCode>69004</ShipPostalCode>
    <ShipCountry>France</ShipCountry>
    </Orders>
  - <OrderDetails diffgr:id="OrderDetails1" msdata:rowOrder="0">
    <OrderID>10250</OrderID>
    <ProductID>41</ProductID>
    <UnitPrice>7.7</UnitPrice>
    <Quantity>10</Quantity>
    <Discount>0</Discount>
    <OrderID1>10250</OrderID1>
    <CustomerID>HANAR</CustomerID>
    <EmployeeID>4</EmployeeID>
    <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
    <RequiredDate>1996-08-05T00:00:00.0000000-07:00</RequiredDate>
    <ShippedDate>1996-07-12T00:00:00.0000000-07:00</ShippedDate>
    <ShipVia>2</ShipVia>
    <Freight>65.83</Freight>
    <ShipName>Hanari Carnes</ShipName>
    <ShipAddress>Rua do Paço, 67</ShipAddress>
    <ShipCity>Rio de Janeiro</ShipCity>
    <ShipRegion>RJ</ShipRegion>
    <ShipPostalCode>05454-876</ShipPostalCode>
    <ShipCountry>Brazil</ShipCountry>
    </OrderDetails>
  - <OrderDetails diffgr:id="OrderDetails2" msdata:rowOrder="1">
    <OrderID>10250</OrderID>
    <ProductID>51</ProductID>
    <UnitPrice>42.4</UnitPrice>
    <Quantity>35</Quantity>
    <Discount>0.15</Discount>
    <OrderID1>10250</OrderID1>
    <CustomerID>HANAR</CustomerID>
    <EmployeeID>4</EmployeeID>
    <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
    <RequiredDate>1996-08-05T00:00:00.0000000-07:00</RequiredDate>
    <ShippedDate>1996-07-12T00:00:00.0000000-07:00</ShippedDate>
    <ShipVia>2</ShipVia>
    <Freight>65.83</Freight>
    <ShipName>Hanari Carnes</ShipName>
    <ShipAddress>Rua do Paço, 67</ShipAddress>
    <ShipCity>Rio de Janeiro</ShipCity>
    <ShipRegion>RJ</ShipRegion>
    <ShipPostalCode>05454-876</ShipPostalCode>
    <ShipCountry>Brazil</ShipCountry>
    </OrderDetails>
  - <OrderDetails diffgr:id="OrderDetails3" msdata:rowOrder="2">
    <OrderID>10250</OrderID>
    <ProductID>65</ProductID>
    <UnitPrice>16.8</UnitPrice>
    <Quantity>15</Quantity>
    <Discount>0.15</Discount>
    <OrderID1>10250</OrderID1>
    <CustomerID>HANAR</CustomerID>
    <EmployeeID>4</EmployeeID>
    <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
    <RequiredDate>1996-08-05T00:00:00.0000000-07:00</RequiredDate>
```

This document is created with the unregistered version of CHM2PDF Pilot


```
  <ShippedDate>1996-07-12T00:00:00.0000000-07:00</ShippedDate>
  <ShipVia>2</ShipVia>
  <Freight>65.83</Freight>
  <ShipName>Hanari Carnes</ShipName>
  <ShipAddress>Rua do Paço, 67</ShipAddress>
  <ShipCity>Rio de Janeiro</ShipCity>
  <ShipRegion>RJ</ShipRegion>
  <ShipPostalCode>05454-876</ShipPostalCode>
  <ShipCountry>Brazil</ShipCountry>
  </OrderDetails>
- <OrderDetails diffgr:id="OrderDetails4" msdata:rowOrder="3">
  <OrderID>10251</OrderID>
  <ProductID>22</ProductID>
  <UnitPrice>16.8</UnitPrice>
  <Quantity>6</Quantity>
  <Discount>0.05</Discount>
  <OrderID1>10251</OrderID1>
  <CustomerID>VICTE</CustomerID>
  <EmployeeID>3</EmployeeID>
  <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
  <RequiredDate>1996-08-22T00:00:00.0000000-07:00</RequiredDate>
  <ShippedDate>1996-07-15T00:00:00.0000000-07:00</ShippedDate>
  <ShipVia>1</ShipVia>
  <Freight>41.5</Freight>
  <ShipName>Victuailles en stock</ShipName>
  <ShipAddress>2, rue du Commerce</ShipAddress>
  <ShipCity>Lyon</ShipCity>
  <ShipPostalCode>69004</ShipPostalCode>
  <ShipCountry>France</ShipCountry>
  </OrderDetails>
- <OrderDetails diffgr:id="OrderDetails5" msdata:rowOrder="4">
  <OrderID>10251</OrderID>
  <ProductID>57</ProductID>
  <UnitPrice>15.6</UnitPrice>
  <Quantity>15</Quantity>
  <Discount>0.05</Discount>
  <OrderID1>10251</OrderID1>
  <CustomerID>VICTE</CustomerID>
  <EmployeeID>3</EmployeeID>
  <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
  <RequiredDate>1996-08-22T00:00:00.0000000-07:00</RequiredDate>
  <ShippedDate>1996-07-15T00:00:00.0000000-07:00</ShippedDate>
  <ShipVia>1</ShipVia>
  <Freight>41.5</Freight>
  <ShipName>Victuailles en stock</ShipName>
  <ShipAddress>2, rue du Commerce</ShipAddress>
  <ShipCity>Lyon</ShipCity>
  <ShipPostalCode>69004</ShipPostalCode>
  <ShipCountry>France</ShipCountry>
  </OrderDetails>
- <OrderDetails diffgr:id="OrderDetails6" msdata:rowOrder="5">
  <OrderID>10251</OrderID>
  <ProductID>65</ProductID>
  <UnitPrice>16.8</UnitPrice>
  <Quantity>20</Quantity>
  <Discount>0</Discount>
  <OrderID1>10251</OrderID1>
  <CustomerID>VICTE</CustomerID>
  <EmployeeID>3</EmployeeID>
  <OrderDate>1996-07-08T00:00:00.0000000-07:00</OrderDate>
  <RequiredDate>1996-08-22T00:00:00.0000000-07:00</RequiredDate>
  <ShippedDate>1996-07-15T00:00:00.0000000-07:00</ShippedDate>
  <ShipVia>1</ShipVia>
  <Freight>41.5</Freight>
  <ShipName>Victuailles en stock</ShipName>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
<ShipAddress>2, rue du Commerce</ShipAddress>
<ShipCity>Lyon</ShipCity>
<ShipPostalCode>69004</ShipPostalCode>
<ShipCountry>France</ShipCountry>
</OrderDetails>
</NewDataSet>
</diffgr:diffgram>
</DataSet>
```

Dig into the XML and look for the tag "- <xsd:unique name="Constraint1">". This starts the section that defines the relationship between the tables. It says that a constraint named Constraint1 defines the relationship between the DataTable named Orders and the DataTable named OrderDetails. The relationship is on a field named OrderID in each DataTable.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Consuming Web Services

Now that we have created several XML Web services, let's take a look at how to consume them. As mentioned earlier, XML Web services can be consumed by any client that is capable of making a request over HTTP and parsing out the returned XML. The .NET framework is capable of working in this fashion, but it also has tools for creating something called a Web service proxy that greatly simplifies access to a Web service. You can create a Web service proxy in two ways. If you are using Visual Studio .NET, you can add what is called a Web Reference by pointing Visual Studio .NET to the URL of the Web service. If you are not using Visual Studio .NET, you can use a tool called Web Service Description Language Tool (wsdl.exe) to create the Web service proxy.

Let's take a look at wsdl.exe first. At a minimum, the utility requires a path to a Web service or to the WSDL that describes the Web service—hence the name of the utility. Given this, it will generate the proxy class. This class has the same method signatures as the Web service and hides the implementation details so that calling the Web service is transparent. If we run wsdl.exe against the SimpleDataSet example with the following command line:

```
Wsdl http://localhost/book/webservices/simpledataset/dataset.asmx /language:cs
```

We get back a new file named after the class contained within the dataset.asmx file, datasetsample.cs. This file is shown in Listing 6.12.

Listing 6.12 A Proxy Class (datasetsample.cs) for SimpleDataSet Generated with the WSDL Tool

```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.2914.16
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------

//
// This source code was auto-generated by wsdl, Version=1.0.2914.16.
//
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.Web.Services;

[System.Web.Services.WebServiceBindingAttribute(Name="DataSetSampleSoap",
Namespace="http://tempuri.org/")]
public class DataSetSample : System.Web.Services.Protocols.SoapHttpClientProtocol {

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public DataSetSample() {
        this.Url =
"http://localhost/book/webservices/csharp/simpledataset/dataset.asmx";
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/
```

This document is created with the unregistered version of CHM2PDF Pilot

```
Simple", Use=System.Web.Services.Description.SoapBindingUse.Literal,
Parameter-Style=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Data.DataSet Simple() {
        object[] results = this.Invoke("Simple", new object[0]);
        return ((System.Data.DataSet)(results[0]));
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public System.IAsyncResult BeginSimple(System.AsyncCallback callback, object
asyncState) {
        return this.BeginInvoke("Simple", new object[0], callback, asyncState);
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public System.Data.DataSet EndSimple(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((System.Data.DataSet)(results[0]));
    }
}
```

This new proxy class can then be included in a project to encapsulate access to the Web service. If we want to use it in a Windows forms project, we can include it in our project. We then use it by creating a new instance of the Web service object as though it is a local object instead of a remote one. Listing 6.13 shows a Windows form with a data grid on it, which retrieves the DataSet from SimpleDataSet and binds it to a form.

Listing 6.13 A Form That Is Bound to the SimpleDataSet Web Service

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SimpleFormsClient
{
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.DataGrid dataGrid1;
        private System.ComponentModel.Container components = null;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
        }

        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
```

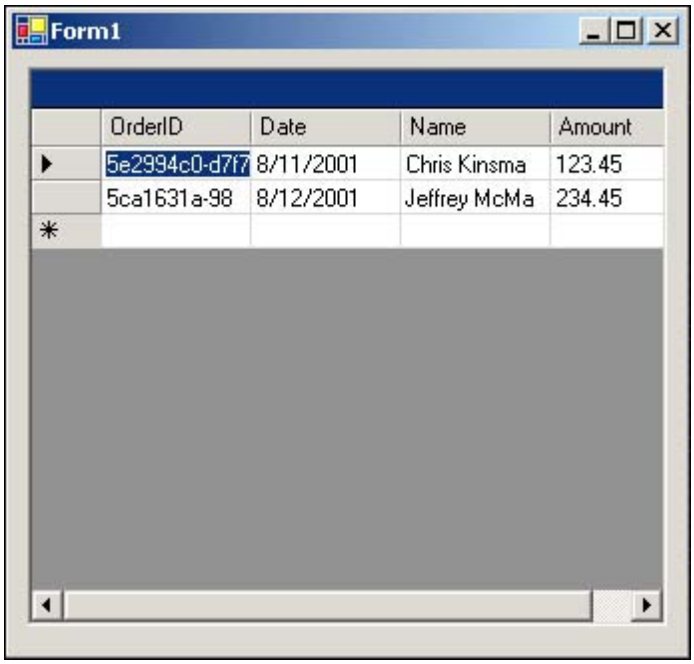
This document is created with the unregistered version of CHM2PDF Pilot

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.dataGrid1 = new System.Windows.Forms.DataGrid();
            ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)) .BeginInit();
            this.SuspendLayout();
            //
            // dataGrid1
            //
            this.dataGrid1.DataMember = "";
            this.dataGrid1.Dock = System.Windows.Forms.DockStyle.Fill;
            this.dataGrid1.Name = "dataGrid1";
            this.dataGrid1.Size = new System.Drawing.Size(504, 389);
            this.dataGrid1.TabIndex = 0;
            //
            // Form1
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(504, 389);
            this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                                                        this.
dataGrid1});
            this.Name = "Form1";
            this.Text = "Form1";
            this.Load += new System.EventHandler(this.Form1_Load);
            ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).EndInit();
            this.ResumeLayout(false);

        }
        #endregion

        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        private void Form1_Load(object sender, System.EventArgs e)
        {
            DataSetSample dss = new DataSetSample();
            dataGrid1.DataMember = "Orders";
            dataGrid1.DataSource = dss.Simple();
        }
    }
}
```

The important stuff is in the last few lines. I have added three lines of code that do all the work to the form load. The first lines get a new instance of the Web service proxy class. Then, calling the WebMethod on the new class is as simple as the last line: dss.Simple(). That's it. The .NET framework hides all the hard stuff, making calling remote Web methods on a Web service as easy as calling methods on local classes. Figure 6.9 shows the resulting form.

Figure 6.9. A Windows form showing the result of calling the SimpleDataSet Web service.

This document is created with the unregistered version of CHM2PDF Pilot



SoapHttpClientProtocol

This is the class from which the Web proxies generated by WSDL and Visual Studio .NET derive.

Of course, you aren't limited to calling XML Web services from Windows forms. It is just as easy to call a Web service from a Web form. This time around, I am going to include a Web reference in a Visual Studio .NET Web form project. I do this by pointing the Add Web Reference dialog box to the URL where the XML Web services resides. The dialog box will automatically find the WSDL and allow me to add the reference. Visual Studio .NET will then create the proxy class for me, eliminating the need for wsdl.exe. Visual Studio .NET names the proxy slightly differently than when you create it with wsdl.exe. The biggest difference is that it creates a namespace that is set to the hostname.domainname combination of the Web service that you created it from. Listing 6.14 shows the proxy that was created by Visual Studio.NET.

Listing 6.14 The Proxy Created for SimpleDataSet by Visual Studio .NET

```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.2914.16
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------

namespace SimpleDataSetWebClient.localhost {
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.Web.Services;
    [System.Web.Services.WebServiceBindingAttribute(Name="DataSetSampleSoap",
Namespace="http://tempuri.org/")]
    public class DataSetSample : System.Web.Services.Protocols. SoapHttpClientProtocol {

        [System.Diagnostics.DebuggerStepThroughAttribute()]
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        public DataSetSample() {
            this.Url =
"http://localhost/book/webservices/csharp/simpledataset/dataset.asmx";
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://
tempuri.org/Simple", Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        public System.Data.DataSet Simple() {
            object[] results = this.Invoke("Simple", new object[0]);
            return ((System.Data.DataSet)(results[0]));
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public System.IAsyncResult BeginSimple(System.AsyncCallback callback,
object asyncState) {
            return this.BeginInvoke("Simple", new object[0], callback, asyncState);
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public System.Data.DataSet EndSimple(System.IAsyncResult asyncResult) {
            object[] results = this.EndInvoke(asyncResult);
            return ((System.Data.DataSet)(results[0]));
        }
    }
}
```

When using a Web form to create this object, we have to use slightly different syntax, which in this case is localhost.DataSetSample. The namespace is fixed, even if you change the location that you use to access the Web service. If you right-click the localhost reference in Visual Studio .NET, you can rename the localhost, which will change the namespace. If you want to change the location that is used to access the Web service, you can use the URL property of the proxy class. This property expects a fully qualified reference to the .ASMX file that matches the proxy class. The Web form client for SimpleDataSet shown in Listing 6.15 uses the URL property to change the location.

Listing 6.15 A Web Form Client for SimpleDataSet

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace SimpleDataSetWebClient
{
    public class WebForm1 : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.DataGrid DataGrid1;

        public WebForm1()
        {
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        {
            localhost.DataSetSample dss = new localhost.DataSetSample();

            dss.Url =
"http://localhost/book/webservices/csharp/SimpleDataSet/DataSet.asmx";

            // Indicate which table in the dataset should be bound to
            DataGrid1.DataMember = "Orders";
            // Get the dataset and set it to the source
            DataGrid1.DataSource = dss.Simple();
            // Force the binding to happen
            DataGrid1.DataBind();
        }

        private void Page_Init(object sender, EventArgs e)
        {
            InitializeComponent();
        }

        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion
    }
}
```

Again, the interesting lines are the ones in Page_Load. The first line creates a new instance of the proxy class using the localhost namespace. The next line changes the URL from the initial one used to create the proxy to the one that will be used in the "production" environment. Then the datagrid binding syntax binds the returned dataset directly to the grid. The last line calls DataBind() to tell the framework that it is now time to perform the binding.

Asynchronous Clients

XML Web services are a convenient way to access services over the Internet. The Internet itself can introduce some uncertainties in calling your XML Web services, however. The latencies involved in transiting data from point A to point B on the Internet change on an hourly basis, if not second to second. You don't want to have your application block or appear to be sluggish because you are retrieving information from a Web service over the Internet. The solution is to call the Web service in an asynchronous fashion. This enables you to fire off the request to a Web service and then continue doing other work. When the Web service request returns, you can retrieve the data and display it to the user.

Asynchronous access is more useful in a Windows form type of application where you can go ahead and make the form available to the user immediately. When the data becomes available, just update it in the already displayed form. The Web service proxy again does the majority of the heavy lifting. In addition to creating mirrors of all the Web methods for the Web service, it creates a Begin<methodname> and End<methodname> method for each Web method.

This document is created with the unregistered version of CHM2PDF Pilot

In the proxy for the SimpleDataSet Web service shown in Listing 6.13, you will see, in addition to the Simple() method, a BeginSimple and EndSimple method. These are already set up to work with the IAsyncResult interface. When the Begin method is called, it expects to be passed, in addition to any arguments the Web method requires, the address of a callback method. A callback method is just a method that is called when the operation completes. Optionally, you can stick any object into the AsyncState parameter and retrieve it later in the callback. This is useful to get a handle on the Web service so that you don't have to store a reference to it in a global variable. You will need this reference to call the End method so that you can retrieve the results from the Web service. Listing 6.16 shows a Web form that calls a new Web service, NorthwindOrder, that utilizes this methodology.

Listing 6.16 A Windows Form That Calls the New NorthwindOrder Web Service Asynchronously

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace NorthwindFormsClient
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        public delegate void SetData(IAsyncResult ar);

        private System.Windows.Forms.DataGrid dataGrid1;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.dataGrid1 = new System.Windows.Forms.DataGrid();
            ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).
BeginInit();
            this.SuspendLayout();
            //
            // dataGrid1
            //
            this.dataGrid1.DataMember = "";
            this.dataGrid1.Dock = System.Windows.Forms.DockStyle.Fill;
            this.dataGrid1.HeaderForeColor = System.Drawing.SystemColors.\
ControlText;
            this.dataGrid1.Name = "dataGrid1";
            this.dataGrid1.Size = new System.Drawing.Size(552, 429);
            this.dataGrid1.TabIndex = 0;
            //
            // Form1
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(552, 429);
            this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                                                        this.
dataGrid1});
            this.Name = "Form1";
            this.Text = "Form1";
            this.Load += new System.EventHandler(this.Form1_Load);
            ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).EndInit();
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        private void Form1_Load(object sender, System.EventArgs e)
        {
            localhost.Orders oOrders = new localhost.Orders();

            // Create the callback to pass to the asynchronous invocation
            AsyncCallback wscb = new AsyncCallback(this.WebServiceCallback);
            //  Call the web method asynchronously passing in the callback and the
service itself
            oOrders.BeginGetAllOrders(wscb, oOrders);
        }

        public void WebServiceCallback(IAsyncResult ar)
        {
            // When this callback executes we are on a different thread than the
grid
            // Windows Forms is single threaded so we need to call invoke to cross
threads
            SetData dlg = new SetData(SetDataInGrid);
            dataGrid1.Invoke(dlg, new Object[] {ar});
        }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        public void SetDataInGrid(IAsyncResult ar)
        {

            localhost.Orders oOrders;

            // Grab the web service out of the async result object AsyncState
property
            oOrders = (localhost.Orders)ar.AsyncState;
            // Get the data out of the finished web service
            DataSet ds = oOrders.EndGetAllOrders(ar);

            // Put the data into the grid
            dataGrid1.DataMember = "Orders";
            dataGrid1.DataSource = ds;
        }
    }
}
```

Don't get confused by the invoke in WebServiceCallback. Windows forms are single threaded by nature. When the callback from the Web service fires, you are not on the thread that created the control. If you attempt to set the DataSource property while on the wrong thread, you can cause undesirable results, including your program hanging. The invoke is used to transfer control to the thread that created the datagrid and then load the data on that thread.

Asynchronous calls are harder in a Web page than in a Windows form. After a Web page has been sent back to the browser, there is no way to update information in it further. Asynchronous calls are still of limited use in a Web page, however. If you have several Web service calls to make to create a page, fire them all off in an asynchronous fashion at the start of page processing and then continue doing other work in the page—perhaps retrieving information from a database, performing calculations, or doing anything else required to build the page.

This brings us to the other ways of calling a Web service asynchronously. It is possible to call the Web method using Begin but without specifying a callback method. You can then continue with other processing. When you need the data from the Web service, you have two options:

1.

1. Loop while looking at the IsCompleted property of the AsyncResult object. If all you are doing in the loop is checking the IsCompleted property, this is not the most efficient technique. It has the disadvantage of chewing up CPU cycles that other processes could be using. It has the advantage, however, of letting you do other work while waiting for the Web service to finish its work.

1.
2.

2. Utilize the AsyncWaitHandle of the AsyncResult object to cause the thread to wait until the Web service signals completion. This doesn't spin the CPU, wasting needless processing cycles. eeYou can specify a timeout for the wait and then check the IsCompleted property to see if a timeout has occurred. The disadvantage of this, however, is that your code can't be off doing other processing while waiting for the call to return.

2.

Listing 6.17 shows an example of a Web form calling the NorthwindOrders Web service asynchronously.

Listing 6.17 A Web Form That Calls the NorthwindOrders Service Asynchronously and Loads the Orders

This document is created with the unregistered version of CHM2PDF Pilot

into a Grid

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace NorthwindWebClient
{
    /// <summary>
    /// Summary description for WebForm1.
    /// </summary>
    public class WebForm1 : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.DataGrid DataGrid1;
        private void Page_Load(object sender, System.EventArgs e)
        {
            IAsyncResult ar;
            localhost.Orders oOrders = new localhost.Orders();

            // Start the web service call
            ar = oOrders.BeginGetAllOrders(null, null);

            // Do other work....

            // All done so wait for the web service to come back
            // This waitone waits for 20 seconds and then continues
            ar.AsyncWaitHandle.WaitOne(20000, false);

            // Check to see if the async call completed.
            // If not write a timeout message
            if(!ar.IsCompleted)
                Response.Write("Timed out");
            else
            {
                // Data is ready so put it into the grid
                DataGrid1.DataMember = "Orders";
                DataGrid1.DataSource = oOrders.EndGetAllOrders(ar);
                DataGrid1.DataBind();
            }
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion
    }
}
```

This code fires off the asynchronous Web method right at the beginning of page load. It then continues to do other processing. Just before rendering the page back to the user, it waits for the results from the WebMethod for 20 seconds. If the Web method completes sooner, WaitOne exits as soon as the method completes. This means that WaitOne will wait at most 20 seconds but may wait for as few as 0 seconds.

Cookies and Proxies

By default, the proxies created by WSDL or Visual Studio .NET do not interact with cookies. This means that even though you may turn on Session state in the Web service, unless you take a few extra steps on the client, you will never get persistent Session state.

SoapHttpClientProtocol has a CookieContainer property, which is intended to hold a reference to the cookie container class that can be used to maintain cookie information across invocations of Web methods. By default, this property is empty. It is quite easy, however, to create a new cookie container and put a reference to it into the property. Either the Web service reference or the cookie container must persist across invocations, most likely as a member of your top-level class for this to work. Listing 6.18 shows an example of a Windows form that creates a cookie container and puts it into the Web proxy. It utilizes the state Web service that we created back in Listing 6.5.

Listing 6.18 A Windows Form That Creates a Cookie Container and Utilizes the State Web Service

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace StateClient
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        internal System.Windows.Forms.Button btnGet;
        internal System.Windows.Forms.Label lblGetValueText;
        internal System.Windows.Forms.Label lblGetValue;
        internal System.Windows.Forms.TextBox txtGetKey;
        internal System.Windows.Forms.Label lblGetKey;
        internal System.Windows.Forms.Button btnSet;
        internal System.Windows.Forms.TextBox txtSetValue;
        internal System.Windows.Forms.TextBox txtSetKey;
        internal System.Windows.Forms.Label lblSetValue;
        internal System.Windows.Forms.Label lblSetKey;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        localhost.State ss = new localhost.State();
```

This document is created with the unregistered version of CHM2PDF Pilot

```
public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();

    // Initialize the cookie container and set it so we can
    // maintain state
    ss.CookieContainer = new System.Net.CookieContainer();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.btnGet = new System.Windows.Forms.Button();
    this.lblGetValueText = new System.Windows.Forms.Label();
    this.lblGetValue = new System.Windows.Forms.Label();
    this.txtGetKey = new System.Windows.Forms.TextBox();
    this.lblGetKey = new System.Windows.Forms.Label();
    this.btnSet = new System.Windows.Forms.Button();
    this.txtSetValue = new System.Windows.Forms.TextBox();
    this.txtSetKey = new System.Windows.Forms.TextBox();
    this.lblSetValue = new System.Windows.Forms.Label();
    this.lblSetKey = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // btnGet
    //
    this.btnGet.Location = new System.Drawing.Point(128, 112);
    this.btnGet.Name = "btnGet";
    this.btnGet.TabIndex = 19;
    this.btnGet.Text = "Get";
    this.btnGet.Click += new System.EventHandler(this.btnGet_Click);
    //
    // lblGetValueText
    //
    this.lblGetValueText.Location = new System.Drawing.Point(16, 168);
    this.lblGetValueText.Name = "lblGetValueText";
    this.lblGetValueText.Size = new System.Drawing.Size(100, 16);
    this.lblGetValueText.TabIndex = 18;
    //
    // lblGetValue
    //
    this.lblGetValue.Location = new System.Drawing.Point(16, 152);
```

This document is created with the unregistered version of CHM2PDF Pilot

```
this.lblGetValue.Name = "lblGetValue";
this.lblGetValue.Size = new System.Drawing.Size(88, 16);
this.lblGetValue.TabIndex = 17;
this.lblGetValue.Text = "Value:";
//
// txtGetKey
//
this.txtGetKey.Location = new System.Drawing.Point(16, 128);
this.txtGetKey.Name = "txtGetKey";
this.txtGetKey.TabIndex = 16;
this.txtGetKey.Text = "Key";
//
// lblGetKey
//
this.lblGetKey.Location = new System.Drawing.Point(16, 112);
this.lblGetKey.Name = "lblGetKey";
this.lblGetKey.Size = new System.Drawing.Size(72, 16);
this.lblGetKey.TabIndex = 15;
this.lblGetKey.Text = "Key:";
//
// btnSet
//
this.btnSet.Location = new System.Drawing.Point(128, 16);
this.btnSet.Name = "btnSet";
this.btnSet.TabIndex = 14;
this.btnSet.Text = "Set";
this.btnSet.Click += new System.EventHandler(this.btnSet_Click);
//
// txtSetValue
//
this.txtSetValue.Location = new System.Drawing.Point(16, 72);
this.txtSetValue.Name = "txtSetValue";
this.txtSetValue.TabIndex = 13;
this.txtSetValue.Text = "Value";
//
// txtSetKey
//
this.txtSetKey.Location = new System.Drawing.Point(16, 32);
this.txtSetKey.Name = "txtSetKey";
this.txtSetKey.TabIndex = 11;
this.txtSetKey.Text = "Key";
//
// lblSetValue
//
this.lblSetValue.Location = new System.Drawing.Point(16, 56);
this.lblSetValue.Name = "lblSetValue";
this.lblSetValue.Size = new System.Drawing.Size(88, 16);
this.lblSetValue.TabIndex = 12;
this.lblSetValue.Text = "Value:";
//
// lblSetKey
//
this.lblSetKey.Location = new System.Drawing.Point(16, 16);
this.lblSetKey.Name = "lblSetKey";
this.lblSetKey.Size = new System.Drawing.Size(72, 16);
this.lblSetKey.TabIndex = 10;
this.lblSetKey.Text = "Key:";
//
// Form1
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(216, 189);
this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                                                this.
```

This document is created with the unregistered version of CHM2PDF Pilot

```
btnGet,
                                                                        this.
lblGetValueText,
                                                                        this.
lblGetValue,
                                                                        this.
txtGetKey,
                                                                        this.
lblGetKey,
                                                                        this.
btnSet,
                                                                        this.
txtSetValue,
                                                                        this.
txtSetKey,
                                                                        this.
lblSetValue,
                                                                        this.
lblSetKey});
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        private void btnSet_Click(object sender, System.EventArgs e)
        {
            // Set the value
            ss.SetValue(this.txtSetKey.Text, this.txtSetValue.Text);
        }

        private void btnGet_Click(object sender, System.EventArgs e)
        {
            // Get the value
            this.lblGetValueText.Text = ss.GetValue(this.txtGetKey.Text);
        }
    }
}
```

In this form, the Web service is a member variable of Form1. It persists for the life of the form. On form load, a cookie container is created and associated with the instance of the Web service. This enables Session state to work across each of the Web service method calls.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Class Reference

This section provides a quick interface reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book, so for the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

WebService

Member of System.Web.Services.

Assembly: System.Web.Services.dll.

The WebService class is what all XML Web services derive from.

Properties		
Application	Context	Server
Session	User	

WebMethodAttribute

Member of System.Web.Services.

Assembly: System.Web.Services.dll.

The WebMethodAttribute class is used to set the options for a Web method.

Properties		
BufferResponse	CacheDuration	Description
EnableSession	MessageName	TransactionOption
TypeID		

SoapHttpClientProtocol

Member of System.Web.Services.Protocols.HttpWebClientProtocol.

This document is created with the unregistered version of CHM2PDF Pilot

Assembly: System.Web.Services.dll.

This is the class from which the Web proxies generated by WSDL and Visual Studio .NET derive.

Properties		
AllowAutoRedirect	ClientCertificates	ConnectionGroupName
CookieContainer	Credentials	PreAuthenticate
Proxy	RequestEncoding	Timeout
Url	UserAgent	
Methods		
Abort	Discover	

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 7. Security

IN THIS CHAPTER

- Identity and Principal
- Windows Authentication
- Forms Authentication
- Passport Authentication
- File Authorization
- URL Authorization
- Custom Roles with Forms Authentication
- Pulling It All Together
- Impersonation
- Class Reference

ASP.NET offers a number of ways to secure your Web application. Securing a Web application usually breaks down to two tasks: authentication and authorization.

This document is created with the unregistered version of CHM2PDF Pilot

Authentication is the process of determining who the user is. This is frequently done by requiring users to first indicate who they are by providing a name or e-mail address. Second, users are frequently required to provide some shared secret, which theoretically only they know. The most common example of a shared secret is a password. The combination of the name and shared secret is then compared against some store containing user information. This combination of a username and password is fre-quently referred to as a set of credentials. If the provided credentials match the information in the store, the user is deemed authentic and is allowed access to the application. If the information does not match, the user is typically given another chance to provide valid credentials. ASP.NET includes three implementations of authentication schemes: Windows, Forms, and Passport.

The other task, authorization, is the process of determining what resources users should have access to after they have been authenticated. This process is typically performed by comparing a list of roles applicable to the authenticated user against a list of roles that are required for access to a particular resource. These resources could be Web pages, graphics, or pieces of information from a database. ASP.NET includes two implementations of authorization schemes: file and URL.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Identity and Principal

Two key objects closely associated with security in .NET are identities and principal objects. An identity represents a user's identity and the method used to authenticate the user. Two identity classes are provided by default in .NET. The GenericIdentity is a generic implementation of the IIdentity interface that is not specific to any particular type of authentication. It implements the required interface and no more. The WindowsIdentity is an implementation of IIdentity that adds more methods and properties particular to Windows-based authentication.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Class Reference

This section provides a quick interface reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book; for the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

GenericIdentity Class

Member of System.Security.Principal.

Assembly: mscorlib.dll.

The GenericIdentity class represents the identity of authenticated users and the method used to authenticate them. Derive from this class to provide additional information to the Identity specific to a custom authentication method.

Properties		
AuthenticationType	IsAuthenticated	Name

WindowsIdentity Class

Member of System.Security.Principal.

Assembly: mscorlib.dll.

The WindowsIdentity class is used when you want to rely on Windows security. This class implements IIdentity and adds additional properties dealing with Windows security.

Properties		
AuthenticationType	IsAnonymous	IsAuthenticated
IsGuest	IsSystem	Name
Token		
Methods		
GetAnonymous	GetCurrent	Impersonate

The IPrincipal interface represents the security context of a user. The principal contains an Identity for the user as well as any role-based information about the user. Just as .NET provided two implementations of IIdentity, it also provides two implementations of IPrincipal. The first, GenericPrincipal, is a generic implementation of IPrincipal with

This document is created with the unregistered version of CHM2PDF Pilot

a simple array of strings representing the roles. The roles are passed in as part of the constructor. It also provides an implementation of IsInRole() that checks whether a passed-in role is contained within the list of within the list of roles.

GenericPrincipal Class

Member of System.Security.Principal.

Assembly: mscorlib.dll.

The GenericPrincipal class is a minimal implementation of the IPrincipal interface.

Properties
Identity
Methods
IsInRole

WindowsPrincipal Class

Member of System.Security.Principal.

Assembly: mscorlib.dll.

The WindowsPrincipal class is an implementation of IPrincipal intended for use with Windows-based groups. The implementation of IsInRole() checks the user's membership in a Windows local or domain group.

Properties
Identity
Methods
IsInRole

WindowsBuiltInRole Enumeration

Member of System.Security.Principal.

Assembly: mscorlib.dll.

The WindowsBuiltInRole enumeration provides a language-independent way to check for membership in the built-in Windows groups.

This document is created with the unregistered version of CHM2PDF Pilot

Values		
AccountOperator	Administrator	BackupOperator
Guest	PowerUser	PrintOperator
Replicator	SystemOperator	User

PassportIdentity

Member of System.Web.Security.

Assembly: System.Web.dll.

This class contains functionality for interacting with the passport authentication service.

Properties		
AuthenticationType	Error	GetFromNetworkServer
HasSavedPassword	HasTicket	IsAuthenticated
Item	Name	TicketAge
TimeSinceSignIn		
Methods		
AuthUrl2	GetDomainAttribute	GetDomainFromMemberName
GetIsAuthenticated	GetProfileObject	HasFlag
HasProfile	HaveConsent	LoginUser
LogoTag2		
Static Methods		
Compress	CryptIsValid	CryptPutHost
CryptPutSite	Decompress	Decrypt
Encrypt	SignOut	

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Windows Authentication

ASP.NET still requires Internet Information Server (IIS) to handle Web requests. ASP.NET is layered on top of IIS using an ISAPI filter just like ASP.old. What this means is that ASP.NET participates in the IIS security model.

Before ASP.NET is even called on to execute a page, IIS must be satisfied that the user has permission to request the page. This permission check is done using any of the standard mechanisms built in to IIS, including Basic Authentication, Digest Authentication, or Integrated Windows Authentication.

When the user first requests a page that requires authentication, IIS initially returns an HTTP 1.1 401 Access Denied error to the browser. Included in the response is the WWW-Authenticate header, which indicates that the browser should collect user credentials and include them with the next request. After Internet Information Server receives the credentials, they are authenticated against the account database and, if they match, the page is executed.

ASP.NET allows the developer to further interact with these built-in Windows-based authentication mechanisms through the use of the WindowsPrincipal and WindowsIdentity classes mentioned earlier.

By default, when you create a Web Application using Visual Studio .NET or even using the Internet Services Manager, anonymous access is enabled by default for the Web site. To force IIS to authenticate all requests aimed at a directory, you must disable anonymous authentication. This will cause IIS to authenticate the user against the Windows account database.

To force ASP.NET to do its part, you must change an entry in the application's web.config file. Specifically, the authentication section must be set to Windows as follows:

```
<authentication mode="Windows" />
```

With this setting in place, ASP.NET will create a WindowsPrincipal object for each authenticated request that it receives and will populate it with a WindowsIdentity. The groups that the user belongs to will also be loaded into the principal, allowing IsInRole() to test for role membership. The username that is placed into the WindowsIdentity will be of the form DOMAIN\UserName. The groups that IsInRole() checks for are of the form DOMAIN\Group, with the exception of built-in groups such as Administrator. Built-in groups are of the form BUILTIN\Administrator, or alternatively, you can use the WindowsBuiltInRole enumeration.

WindowsBuiltInRole Enumeration

Listings 7.1 and 7.2 show a page that is executed after the user is authenticated. It uses the WindowsPrincipal object to

-

This document is created with the unregistered version of CHM2PDF Pilot

- Check whether the user is authenticated
-
-

- Get the username
-
-

- Get the authentication method
-
-

- Check whether the user is an administrator
-

Listing 7.1 ASP.NET Page That Utilizes the WindowsPrincipal Object to Obtain Information About the User

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Security.Principal;

namespace Windows.Administrator
{
    /// <summary>
    /// Summary description for CDefault.
    /// </summary>
    public class CDefault : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.Label lblIsAuthenticated;
        protected System.Web.UI.WebControls.Label lblAuthenticationType;
        protected System.Web.UI.WebControls.Label lblUserName;
        protected System.Web.UI.WebControls.Label lblAdministrator;
        public CDefault()
        {
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
        WindowsPrincipal wp = (WindowsPrincipal) HttpContext.Current.User;
        // Check if the user is authenticated
        lblIsAuthenticated.Text = wp.Identity.IsAuthenticated.ToString();
        // Output the authentication type
        lblAuthenticationType.Text = wp.Identity.AuthenticationType.ToString();
        // Output the user name
        lblUserName.Text = wp.Identity.Name;
        // Is the user an administrator?
        lblAdministrator.Text =
wp.IsInRole(WindowsBuiltInRole.Administrator).ToString();
        }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
        }

        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion
    }
}
```

Listing 7.2 Class File for ASP.NET Page in Listing 7.1

```
<%@ Page language="c#" Codebehind="Default.aspx.cs" AutoEventWireup="false"
Inherits="Windows.Administrator.CDefault" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="CDefault" method="post" runat="server">
            <P>
                I am an Administrator
            </P>
            <P>
                IsAuthenticated:
                <asp:Label id="lblIsAuthenticated" runat="server"></asp:Label>
            </P>
            <P>
                Authentication Type:
                <asp:Label id="lblAuthenticationType" runat="server"></asp:Label>
            </P>
            <P>
                User Name:
                <asp:Label id="lblUserName" runat="server"></asp:Label>
            </P>
            <P>
                Administrator?
                <asp:Label id="lblAdministrator" runat="server"></asp:Label>
            </P>
        </form>
    </body>
</HTML>
```

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Forms Authentication

The previous section showed how easy it is to use Windows authentication in ASP.NET. ASP.NET provides another security mechanism as well: forms authentication. Why would you want to use it? One reason is because Windows authentication, although easy to use, makes a couple significant assumptions.

Windows Versus Forms Authentication

For one thing, Windows authentication assumes you have a scalable Windows domain implementation already in place. However, this is not always a safe assumption. Many Web site administrators prefer not to go to the trouble of designing, implementing, and maintaining the Active Directory implementation on which domain-based security rests. Others may not have the expertise or budget to figure out how to get Active Directory to scale into the millions of users. Without an Active Directory implementation, you can authenticate against the account database that every Windows 2000 server maintains. However, this approach means that this account database must be replicated in some fashion among servers in a cluster or you are limited to a single server.

Ultimately, what all of this comes down to is that you may want to authenticate users against a credential store other than a Windows 2000 domain. Forms authentication provides one way to do this.

Windows authentication also assumes you want only minimal control over the user interface presented to the user. By default, Windows-based authentication uses a standard browser dialog box to collect the user's credentials. If you want to integrate the form to collect credentials into an existing Web page or provide your own login form, you are out of luck. Forms authentication provides a way for you, the developer, to determine what interface the user receives.

All the advantages of forms authentication are not free, however. First, forms authentication requires that the user has cookies enabled. Although ASP.NET has provided a way to track Session state without cookies, it has not provided a way to track forms authentication without cookies. Hopefully, this will come in a future version of ASP.NET. Second, you, the developer, need to create a login page and write some code to make this all work. ASP.NET provides the infrastructure, but you need to provide the specific implementation.

Other Advantages

Maybe you already do this type of authentication. So what's the big deal with forms authentication? Perhaps the most common security mechanism in place today among ASP developers provides many of the same advantages. I provide a customized login page for my users and authenticate them against my credential store. After they are authenticated, I either write a cookie or save their authentication into a session variable. In every page, I have an include file that looks for the Session() value or cookie. If it isn't there, I redirect the user back to the login page. This can be very effective but it has two big problems:

1.

1. What if I forget the include file?

This document is created with the unregistered version of CHM2PDF Pilot

1.
2.

2. How do I protect PDF, ZIP, or JPG files? There is no place to put the code!
2.

Forms authentication enables me to do all this without having to include code in every page to check whether the user was properly authenticated.

NOTE

ASP.NET authentication and authorization is applied only to files that are mapped to the ASP.NET ISAPI filter. This means that, by default, it will not be applied to any file that is loaded, other than the built-in ASP.NET file types listed in Chapter 2, "Page Framework"—for example, a JPG or a ZIP file. If you add these file types to the ISAPI filter, they can participate in the security model.

Process

When forms authentication is enabled and a request is made for a page, ASP.NET first determines whether authentication is needed. If it is, ASP.NET then checks for an authentication cookie in the request. If it is not present, ASP.NET redirects the user to the login page and passes the URL of the original page as a query string parameter, named ReturnURL, to the login page.

NOTE

The user is sent to the login page using a 302 location-moved redirection. This means that any form data that may have been included with the request is lost.

The login page collects users' credentials and is responsible for validating them against a credential store. This is where you as the developer get control. The credential store could consist of an LDAP directory, a database, or even something as simple as an XML file. When the credentials have been verified, RedirectFromLoginPage() is called to write an authentication ticket into a cookie and redirect the user to the original content that the user requested. A diagram of this process is shown in Figure 7.1.

Figure 7.1. A process flow for the forms authentication process.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Passport Authentication

Forms authentication is cool, but it has one potential failing. It requires users to create and maintain a set of credentials for every Web site that they visit. Wouldn't it be nice to have just one username and password that you could use at any site? That is the idea behind Microsoft Passport.

The first step to using Microsoft Passport is to request a Site ID. The Site ID identifies your site as a valid Passport partner and is used as a key by Passport to identify settings related to your site. When you have acquired a Site ID, you can configure your site to use Passport. Change the mode attribute of <authentication> element to Passport. A second optional element <passport> enables you to specify the location of the Passport login page via the redirectUrl attribute. A sample web.config for Passport authentication is shown in Listing 7.6.

Listing 7.6 A web.config Setup for Passport Authentication

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <compilation defaultLanguage="c#" debug="true" />
        <authentication mode="Passport">
            <passport redirectUrl="login.aspx" />
        </authentication>
        <authorization>
            <allow users="*" /> <!-- Allow all users -->
        </authorization>
    </system.web>
    <location path="attendees">
        <system.web>
            <authorization>
            <deny users="?" />
            </authorization>
        </system.web>
    </location>
</configuration>
```

The rest of the web.config is very similar to what you used before with forms authentication. Now you need to provide a way for the user to log in. This is typically done with most participating Passport sites using a Passport logo that indicates whether the user is logged in or out. The PassportIdentity object provides a method LogoTag2() that returns the HTML necessary to display this logo.

PassportIdentity

This class contains functionality for interacting with the Passport authentication service.

If the user is not logged in, however, no PassportIdentity object is available. If User.Identity.IsAuthenticated returns false, you need to create a new instance of PassportIdentity and use it to output the login logo. This logo is frequently placed in a number of locations around a site, so it is a good idea to wrap it up in a Web user control. Listing 7.7 shows an example of this control.

This document is created with the unregistered version of CHM2PDF Pilot

Listing 7.7 A Web User Control That Wraps the Display of the Passport Logo

```
namespace Passport
{
    using System;
    using System.Data;
    using System.Drawing;
    using System.Web;
    using System.Web.UI.WebControls;
    using System.Web.UI.HtmlControls;

    /// <summary>
    ///        Summary description for passportlogo.
    /// </summary>
    public abstract class passportlogo : System.Web.UI.UserControl
    {


        /// <summary>
        public passportlogo()
        {
            this.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
            System.Web.Security.PassportIdentity pi;

        if(HttpContext.Current.User.Identity.IsAuthenticated)
            pi = (System.Web.Security.PassportIdentity)
HttpContext.Current.User.Identity;
        else
            pi = new System.Web.Security.PassportIdentity();

        if(Request["ReturnURL"] == null)
            Response.Write(pi.LogoTag2("http://" +
Request.ServerVariables["SERVER_NAME"].ToString() +
HttpContext.Current.Request.Path, 600, false, "", 1033, false, "", 0, false));
        else
            Response.Write(pi.LogoTag2(Request["ReturnURL"], 600, false, "", 1033,
false, "", 0, false));

        }

        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
        }

        #region Web Form Designer generated code
        ///        Required method for Designer support - do not modify
        ///        the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);
        }
        #endregion
    }
}
```

This document is created with the unregistered version of CHM2PDF Pilot

The user control first looks to see whether the user has been authenticated. If he is authenticated, the user control then grabs the PassportIdentity of the user. If he is not authenticated, a new PassportIdentity is created. The LogoTag2 method of the PassportIdentity object is then used to output the appropriate login or logout logo for Passport.

This user control can now be placed on the home page to provide a way to log in. When a user attempts to access a page for which he has insufficient permissions, he is redirected to the URL indicated in the redirectUrl attribute of the <passport> element. This page should show a message indicating to the user that he attempted to access authenticated content and provide a way to log in. The easiest way to do this is to include the passportlogo user control that you created in Listing 7.7. A sample login.aspx page is shown in Listing 7.8.

Listing 7.8 The login.aspx Page That Is Shown When Users Attempt to Access Content When They Are Not Authenticated

```
<%@ Page language="c#" Codebehind="login.aspx.cs" AutoEventWireup="false"
Inherits="Passport.login" %>
<%@ Register TagPrefix="uc1" TagName="passportlogo" Src="passportlogo.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="login" method="post" runat="server">
            <P>
                You must login to access that content... Click on the passport logo
to log in...
            </P>
            <P>
                 
                <uc1:passportlogo id="Passportlogo1" runat="server">
                </uc1:passportlogo>
            </P>
        </form>
    </body>
</HTML>
```

If you want to use the personalization features of Microsoft Passport, you will also be interested in the Profile collection. The GetProfileObject method of the PassportIdentity allows you to retrieve attributes from the user's Passport profile. The valid attributes are listed in Table 7.1.

Table 7.1. Valid Passport Attributes

Attribute Name	Description	Validation Rules
Accessibility	Indicates whether accessibility features should be enabled on participant sites for this user. 0=no; 1=yes.	Must be 0 or 1.

This document is created with the unregistered version of CHM2PDF Pilot

BDay_precision	Defines the precision of the Birthdate attribute.	0, 1, 2, 3, or Null.
Birthdate	Contains the user's birth year or birth date.	Only dates since 12/30/1899 are valid.
City	GeoID that maps to the user's city.	Must be a valid GeoID.
Country	ISO 3166 country code for the user's country.	
Gender	Gender of user.	Must be Null, M, F, or U.
Lang_Preference	LCID of the user's preferred language.	
MemberName	A legacy attribute that no longer contains a sign-in name, but can be used to determine a user's domain. Use DomainFromMemberName().	
Nickname	Friendly name the user would like to be greeted by.	
PreferredEmail	The user's e-mail address.	
PostalCode	Stores the postal code for the United States and other countries, where appropriate.	
ProfileVersion	Represents the version of the user's core profile.	N/A
Region	GeoID that maps to the region within the user's country.	Must be a valid GeoID.
Wallet	Indicates whether the user has established a Passport wallet.	0=no; 1=yes.

To use the profile data, just pass the name of the core attribute you are interested in to the GetProfileAttribute() method. Listing 7.9 shows a page that retrieves the e-mail address and member name of a user.

Listing 7.9 A Page That Displays the PUID, E-mail Address, and Member Name of a Logged-in Passport User

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace Passport
{
    /// <summary>
    /// Summary description for CDefault.
    /// </summary>
    public class CDefault : System.Web.UI.Page
    {
        public CDefault()
        {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
            if(User.Identity.IsAuthenticated)
            {
                System.Web.Security.PassportIdentity pi =
(System.Web.Security.PassportIdentity)User.Identity;
                Response.Write("<BR>");
                Response.Write("User.Identity.Name: " + User.Identity.Name);
                Response.Write("<BR>");
                Response.Write("Preferred Email: " +
pi["preferredemail"].ToString());
                Response.Write("<BR>");
                Response.Write("Member name: " + pi["membername"].ToString());
                Response.Write("<BR>");
            }

        }

        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
        }
        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);
        }
        #endregion
    }
}
```

When using Passport authentication, User.Identity.Name does not return the login name of the user; instead, it returns a Passport User ID (PUID). You should use this as a key to any user-specific data you are storing, rather than using the member name or e-mail address (which can change) from the profile.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

File Authorization

Now that you know how users are authenticated, you need to control what resources they have access to. File authorization is the method of authorization that existed in ASP.old and migrates forward in ASP.NET.

NOTE

File authorization is the only method of authorization that IIS supports. If Allow Anonymous is off in IIS, authorization is handled by IIS. If Allow Anonymous is on, ASP.NET applies File Authorization.

File Authorization utilizes Access Control Lists (ACLs) to specify the roles that a user must be in to access a file. After the user has been authenticated, the File Authorization module will take each group in the ACL for a file and call IsInRole() on the passing of the name of the group as a parameter to IsInRole(). If IsInRole() returns true, the user is permitted to access the resource. If the File Authorization module iterates across each group in the ACL and never receives a true response from IsInRole(), the user is asked to authenticate again.

File Authorization will work with any type of authentication as long as the principal is populated with the applicable roles as part of the authentication process.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

URL Authorization

One of the downsides of File Authorization is the maintenance of the ACLs on the resources that you want to protect. ACLs are frequently lost when using FTP to transfer files to a Web site. ACLs also have a sometimes-confusing inheritance model that, for a large site, can cause maintenance headaches. ASP.NET provides URL Authorization as a way to authorize users by attaching role information to URLs within a Web site. This URL to Role mapping is done in the web.config file. The URL Authorization module uses the <authorization> element to store this mapping. You saw this earlier in the web.config file used in Listing 7.3. The authorization element can contain both allow and deny elements. Both allow and deny elements have a users attribute, a roles attribute, and a verb attribute. Each attribute will accept a comma-separated list of items. The users attribute is used to match identities. There are two special identities. The "*" identity is used to represent all users. The "?" is used to represent anonymous users.

Before evaluating the authorization element, a merged authorization list is created by taking the authorization list from all of the applicable .config files, including the machine.config file. This merged list is then evaluated from the top to the bottom. Let's look at some examples.

```
<authorization>
    <deny users="?"/>
</authorization>
```

This is a fragment from the web.config in Listing 7.3. What this says is that I want to deny all anonymous users—"?" means anonymous users. But if I am denying all anonymous users, don't I have to allow authenticated users? As it turns out, no. Remember that the preceding list is merged with machine.config at the very least. Machine.config by default contains the following fragment:

```
<authorization>
    <allow users="*"/>
</authorization>
Which means that the merged list during evaluation will look like:
<authorization>
    <deny users="?" />
    <allow users="*" />
</authorization>
```

The first deny users makes sure that no anonymous users are allowed in. The second allow users says that everyone is allowed, which means everything other than anonymous users in this case, meaning only authenticated users. Let's take a look at another fragment.

```
<authorization>
    <allow roles="BUILTIN\Administrators" />
    <deny users="*" />
</authorization>
```

What does this do? Can you guess? Remember top to bottom evaluation. The first element says that if the Principal is a member of the built-in Windows NT group Administrators, I should be allowed access. If I am not a member of this group, the next element takes effect, denying anyone else. You may wonder why the <allow users="*"/> in machine.config doesn't still let everyone in. When the list is merged, conflicting rules are ordered based on which one

This document is created with the unregistered version of CHM2PDF Pilot

is contained within a .config file that is "closer" to the page being requested. Because the web.config is "closer" to the page than the machine.config, the <deny users="*"/> element overrides the <allow users="*"/>. Let's try one more:

```
<authorization>
    <allow users="sa" roles="BUILTIN\Administrators, KINSMAN\Publishers" />
    <deny users="*" />
</authorization>
```

In this case, the one allow element is doing a lot of work. It says that if I am logging in with an account called "sa" or a Principal with the "BUILTIN\Administrators" role or a Principal with the "KINSMAN\Publishers" role, I am granted access. Anyone else is denied access.

<Location>

The authorization element that we have looked at so far applies to the Web site as a whole. In many cases I might want to apply different authorization elements to different parts of the URI namespace. The location element provides a way to do this. The location element allows the developer to specify a different set of settings for a subset of the URI namespace. The location element has two attributes. The first, path, indicates what part of the namespace the settings apply to. The second, allowOverride, can be set to false to prevent another web.config lower in the hierarchy from overriding the settings contained within the location element.

Let's take a Web site that has a need to secure three paths. The http://www.deeptraining.com/attendees path should be available only to people in the attendee, publisher, or administrator roles. The http://www.deeptraining.com/publish path should be available only to people in the publisher or administrator roles. Finally, the path http://www.deeptraining.com/admin should be available only to users in the administrator role. The web.config in Listing 7.10 shows a way to do this using the location element.

Listing 7.10 A web.config That Uses the Location Element to Specify Different Authorization Requirements for Different Directories

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <authentication mode="Forms">
            <forms name="FORMURL" loginUrl="login.aspx" protection="All"
timeout="30" />
        </authentication>
        <authorization>
            <allow users="*" />
        </authorization>
    </system.web>
    <location path="admin">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    <location path="publish">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
                <allow roles="Publisher" />
```

This document is created with the unregistered version of CHM2PDF Pilot

```
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    <location path="attendees">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
                <allow roles="Publisher" />
                <allow roles="Attendees" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
</configuration>
```

Note that I allow all users to access the root of the application. At the very least, I must allow all users to access the login.aspx form, which happens to reside in the root. If you don't want users to be able to access the root in an unauthenticated fashion, create a location element specifically to allow access to login.aspx, like this:

```
<location path="login.aspx">
    <system.web>
        <authorization>
            <allow users="*" />
        </authorization>
    </system.web>
</location>
```

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Custom Roles with Forms Authentication

We have discussed so far how to authenticate a user with forms authentication. This process gives you an identity to authorize against. With Windows authentication, you have both an identity and a list of roles to authorize against. How do you use role mappings with forms authentication to authorize against?

With a little work, you can add roles to the forms authentication model. You will use the web.config in Listing 7.11, which defines the roles that are required for access to various URLs in the application.

Listing 7.11 Web.config That Uses Location Mapping and URL Authorization to Map Roles to Application Locations

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <authentication mode="Forms">
            <forms name="FORMURL" loginUrl="login.aspx" protection="All"
timeout="30" />
        </authentication>
        <!--  AUTHORIZATION
          This section sets the authorization policies of the application. You can
allow or deny access
          to application resources by user or role. Wildcards: "*" mean everyone,
"?" means anonymous
          (unauthenticated) users.
    -->
        <authorization>
            <allow users="*" /> <!-- Allow all users -->
            <!--  <allow     users="[comma separated list of users]"
                             roles="[comma separated list of roles]"/>
                  <deny      users="[comma separated list of users]"
                             roles="[comma separated list of roles]"/>
            -->
        </authorization>
    </system.web>
    <location path="Administrator">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    <location path="Publisher">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
                <allow roles="Publisher" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    <location path="User">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
```

This document is created with the unregistered version of CHM2PDF Pilot

```
                <allow roles="User" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
</configuration>
```

Next, create another simple login form. It will collect a set of credentials and allow a user with a username of either "Administrator" or "Chris" into the application. Listings 7.12 and 7.13 show the login page.

Listing 7.12 A Login Page for Gathering Credentials That Will Authenticate Only Two Users: Chris and Administrator

```
<%@ Page language="c#" Codebehind="login.aspx.cs" AutoEventWireup="false"
Inherits="FormAuthorizationURL.login" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body ms_positioning="GridLayout">
        <form id="login" method="post" runat="server">
            <asp:Label id="lblUserName" style="Z-INDEX: 101; LEFT: 36px; POSITION:
absolute; TOP: 43px" runat="server">User Name:</asp:Label>
            <asp:Button id="btnLogin" style="Z-INDEX: 105; LEFT: 290px; POSITION:
absolute; TOP: 40px" runat="server" Text="Login" tabIndex="4"></asp:Button>
            <asp:TextBox id="txtPassword" style="Z-INDEX: 104; LEFT: 119px;
POSITION: absolute; TOP: 74px" runat="server" tabIndex="3"
TextMode="Password"></asp:TextBox>
            <asp:TextBox id="txtUserName" style="Z-INDEX: 102; LEFT: 119px;
POSITION: absolute; TOP: 41px" runat="server" tabIndex="1"></asp:TextBox>
            <asp:Label id="lblPassword" style="Z-INDEX: 103; LEFT: 36px; POSITION:
absolute; TOP: 77px" runat="server" tabIndex="2">Password:</asp:Label>
            <asp:RequiredFieldValidator id="RequiredFieldValidator1"
style="Z-INDEX: 106; LEFT: 44px; POSITION: absolute; TOP: 131px" runat="server"
ErrorMessage="User Name is required."
ControlToValidate="txtUserName"></asp:RequiredFieldValidator>
            <asp:RequiredFieldValidator id="RequiredFieldValidator2"
style="Z-INDEX: 107; LEFT: 46px; POSITION: absolute; TOP: 165px" runat="server"
ErrorMessage="Password is required."
ControlToValidate="txtPassword"></asp:RequiredFieldValidator>
            <asp:Label id="lblMessage" style="Z-INDEX: 108; LEFT: 44px; POSITION:
absolute; TOP: 8px" runat="server" Width="285px" Height="19px"
ForeColor="Red"></asp:Label>
        </form>
    </body>
</HTML>
```

Listing 7.13 The Class File for the Page in Listing 7.12

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace FormAuthorizationURL
{
    /// <summary>
    /// Summary description for login.
    /// </summary>
    public class login : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.Label lblUserName;
        protected System.Web.UI.WebControls.Button btnLogin;
        protected System.Web.UI.WebControls.TextBox txtPassword;
        protected System.Web.UI.WebControls.TextBox txtUserName;
        protected System.Web.UI.WebControls.Label lblPassword;
        protected System.Web.UI.WebControls.RequiredFieldValidator
RequiredFieldValidator1;
        protected System.Web.UI.WebControls.RequiredFieldValidator
RequiredFieldValidator2;
        protected System.Web.UI.WebControls.Label lblMessage;

        public login()
        {
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }

        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
        }

        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.btnLogin.Click += new System.EventHandler(this.btnLogin_Click);
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void btnLogin_Click(object sender, System.EventArgs e)
        {
            switch(txtUserName.Text)
            {
                case "Chris":
                case "Administrator":
                    System.Web.Security.FormsAuthentication.RedirectFromLoginPage
(txtUserName.Text, false);
                    break;
                default:
                    lblMessage.Text = "Invalid User";
                    break;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            }
        }
    }
}
```

This so far is similar to past examples. Now you need to add one more thing to the mix. Before the authorization process begins, an application event is raised, AuthorizeRequest. By handling this event, you can do some extra work before any page is authorized. In this case, you are going to replace the default principal with a new instance of GenericPrincipal, with one important difference. You are going to fill in the role information. This will allow you to then perform authorization against the roles. The constructor for GenericPrincipal allows you to pass a list of roles into it. After creating the new GenericPrincipal, you will replace the default one with the one you have created.

NOTE

The AuthorizeRequest event is fired for every request, including the initial one when the user isn't authenticated, the one that returns in the login form, and so on. It is a good idea to check whether the user is authenticated prior to attempting to perform the role mapping. This is easily done using the IsAuthenticated property of the identity.

Application events are typically handled in the global.asax. Listing 7.14 shows a global.asax that handles the AuthorizeRequest event. The code first checks to make sure the user is authenticated. If she is, it then maps her into roles based on her identity. A new GenericPrincipal is created with the appropriate role information, and that is it!

Listing 7.14 The global.asax Class That Handles the AuthorizeRequest Application Event

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Web;
using System.Web.SessionState;

namespace FormAuthorizationURL
{
    /// <summary>
    /// Summary description for Global.
    /// </summary>
    public class Global : System.Web.HttpApplication
    {

        protected void Application_AuthorizeRequest(object sender,
System.EventArgs e)
        {
            // Make sure the user is authenticated
            if(HttpContext.Current.User.Identity.IsAuthenticated)
            {
                // Map the user to a role based on their identity
                switch(HttpContext.Current.User.Identity.Name)
                {
                    case "Chris":
                        HttpContext.Current.User = new
System.Security.Principal.GenericPrincipal(HttpContext.Current.User.Identity, new
string[] {"Publisher"});
                        break;
                    case "Administrator":
                        HttpContext.Current.User = new
```

This document is created with the unregistered version of CHM2PDF Pilot

```
System.Security.Principal.GenericPrincipal(HttpContext.Current.User.Identity, new
string[] {"Administrator", "Publisher"});
                        break;
                    default:
                        HttpContext.Current.User = new
System.Security.Principal.GenericPrincipal(HttpContext.Current.User.Identity, new
string[] {"User"});
                        break;
                }
            }
        }
    }
}
```

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Pulling It All Together

So far we have looked at each feature in isolation. Let's try to pull together a realistic example that you might be able to use in your work that combines all these concepts. You are going to create a Web site, as mentioned earlier, that contains three authenticated and authorized subdirectories: attendees, publish, and admin. Forms authentication will be used to authenticate the users against a Microsoft SQL Server–based credential store. URL authorization will be used to protect the subdirectories based on role information stored in Microsoft SQL Server. First, you need to create a web.config file that turns on forms authentication and defines the authorization elements for the appropriate subdirectories. Listing 7.15 shows the web.config.

Listing 7.15 Web.config File Sets Authentication to Forms and Defines the URL Authorization Settings for the Three Subdirectories

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <authentication mode="Forms">
            <forms loginUrl="login.aspx" />
        </authentication>
        <authorization>
            <allow users="*" /> <!-- Allow all users -->
        </authorization>
    </system.web>
    <location path="admin">
        <system.web>
            <authorization>
                <allow roles="Administrator" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    <location path="publish">
        <system.web>
            <authorization>
                <allow roles="Administrator,Publisher" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
    <location path="attendee">
        <system.web>
            <authorization>
                <allow roles="Administrator,Publisher,Attendee" />
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
</configuration>
```

This sets up the following restrictions:

-
- The admin directory requires the Administrator role

This document is created with the unregistered version of CHM2PDF Pilot

-
-
- The publish directory accepts either the Administrator or Publisher roles
-
-
- The attendee directory accepts the Administrator, Publisher, or Attendee roles
-

After this structure is in place, you need to create a login page as in the previous examples. The HTML for this login page is similar to the ones we have shown before; however, the code behind it is very different.

In this example, you are storing the roles associated with a user in Microsoft SQL Server. Each time the user comes back to the site after the initial authentication, you need to add the role information to the Principal as shown in earlier examples. Hitting the database on every request just to retrieve the role information is clearly inefficient. You could potentially cache the role information in Session(), but if you are operating in a Web farm, you would have to make sure you are using some form of shared Session state. Remember, however, that each time you authenticate a user, a cookie is sent down and used for future authentications. It appears to be an ideal location to store the role information. As it turns out, the ticket that is stored in the cookie is represented by the FormsAuthenticationTicket class.

FormsAuthenticationTicket

Member of System.Web.Security.

Assembly: System.Web.dll.

The FormsAuthenticationTicket class represents the data that is encrypted and stored in a cookie for use in forms authentication.

Properties		
CookiePath	Expiration	Expired
IsPersistent	IssueDate	Name
UserData	Version	

This class provides a member, UserData, that can be used to store the role information. This member is a string, not a name/value collection as you might expect. During the initial request on retrieving the role information from the database, you will place it into a comma-separated value string and place this string into the UserData member.

NOTE

Remember that the UserData is passed back and forth from the client to the server on potentially every request. You don't want to store a large amount of data in UserData, because it will slow down performance.

This document is created with the unregistered version of CHM2PDF Pilot

During future requests, you will retrieve the role information from the UserData and use the Split() function to break it up into a string array suitable for passing to the GenericPrincipal constructor. One downside of doing this is that you can no longer use the simple RedirectFromLoginPage() function in the Login page. It instead must do all the work to create the ticket, encrypt it, add it to the Response.Cookies collection, and finally redirect the user to the initial page that he requested. Listings 7.16 and 7.17 show login.aspx, which implements all this functionality.

Listing 7.16 The HTML for login.aspx

```
<%@ Page language="c#" Codebehind="login.aspx.cs" AutoEventWireup="false"
Inherits="DBFormURL.login" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body MS_POSITIONING="GridLayout">
        <form id="login" method="post" runat="server">
            <asp:Label id="lblEmail" style="Z-INDEX: 101; LEFT: 8px; POSITION:
absolute; TOP: 8px" runat="server">Email:</asp:Label>
            <asp:TextBox id="txtEmail" style="Z-INDEX: 102; LEFT: 78px; POSITION:
absolute; TOP: 5px" runat="server"></asp:TextBox>
            <asp:Label id="lblPassword" style="Z-INDEX: 103; LEFT: 8px; POSITION:
absolute; TOP: 44px" runat="server">Password:</asp:Label>
            <asp:TextBox id="txtPassword" style="Z-INDEX: 104; LEFT: 78px;
POSITION: absolute; TOP: 39px" runat="server" TextMode="Password"></asp:TextBox>
            <asp:Button id="btnLogin" style="Z-INDEX: 105; LEFT: 249px; POSITION:
absolute; TOP: 6px" runat="server" Text="Login"></asp:Button>
            <asp:RequiredFieldValidator id="rfvEmail" style="Z-INDEX: 106; LEFT:
13px; POSITION: absolute; TOP: 78px" runat="server" ErrorMessage="You must enter an
email address." ControlToValidate="txtEmail"></asp:RequiredFieldValidator>
            <asp:RequiredFieldValidator id="rfvPassword" style="Z-INDEX: 107; LEFT:
13px; POSITION: absolute; TOP: 105px" runat="server" ErrorMessage="You must enter a
password." ControlToValidate="txtPassword"></asp:RequiredFieldValidator>
            <asp:Label id="lblInvalidPassword" style="Z-INDEX: 108; LEFT: 13px;
POSITION: absolute; TOP: 135px" runat="server" ForeColor="Red"
Visible="False">Invalid
password.</asp:Label>
        </form>
    </body>
</HTML>
```

Listing 7.17 The Class for the login.aspx Page in Listing 7.16

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
namespace DBFormURL
{
    /// <summary>
    /// Summary description for login.
    /// </summary>
    public class login : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.Label lblEmail;
        protected System.Web.UI.WebControls.TextBox txtEmail;
        protected System.Web.UI.WebControls.Label lblPassword;
        protected System.Web.UI.WebControls.TextBox txtPassword;
        protected System.Web.UI.WebControls.Button btnLogin;
        protected System.Web.UI.WebControls.RequiredFieldValidator rfvEmail;
        protected System.Web.UI.WebControls.RequiredFieldValidator rfvPassword;
        protected System.Web.UI.WebControls.Label lblInvalidPassword;

        public login()
        {
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }

        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
        }

        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.btnLogin.Click += new System.EventHandler(this.btnLogin_Click);
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion

        private void btnLogin_Click(object sender, System.EventArgs e)
        {
        SqlDataReader sdr;
        // Create a connection
        SqlConnection sc = new SqlConnection(Application["DSN"].ToString());

        // Open the database connection
        sc.Open();

        // Create a command to get the user
        SqlCommand cmd = new SqlCommand("GetUser '" + txtEmail.Text + "', '" +
txtPassword.Text + "'", sc);

        // Execute the command
        sdr = cmd.ExecuteReader();

        // Attempt to read the first record
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            if(sdr.Read())
            {
                // close the datareader
                sdr.Close();
                // Get the list of roles the user is in
                SqlDataReader drRoles;
                SqlCommand cmdRoles = new SqlCommand("GetRoles '" + txtEmail.Text +
"'", sc);
                ArrayList arRoles = new ArrayList();

                // Execute the command
                drRoles = cmdRoles.ExecuteReader();

                // Get a string builder to store the roles in a csv list
                System.Text.StringBuilder bldr = new System.Text.StringBuilder();

                // Loop through the list of roles and get them
                while(drRoles.Read())
                {
                    bldr.Append(drRoles["Role"]);
                    bldr.Append(",");
                }

                // Strip the last comma
                bldr.Remove(bldr.Length - 1, 1);

                // Create an authentication ticket
                // Place a serialized representation of the roles into the
authentication ticket
                System.Web.Security.FormsAuthenticationTicket ticket = new
System.Web.Security.FormsAuthenticationTicket(1, txtEmail.Text, DateTime.Now,
DateTime.Now.AddMinutes(20), false, bldr.ToString());

                // Get the encrypted version of the ticket
                string strEncrypted =
System.Web.Security.FormsAuthentication.Encrypt(ticket);

                // Put it into a cookie
                HttpCookie hc = new HttpCookie(System.Web.Security.
FormsAuthentication.FormsCookieName, strEncrypted);
                hc.Expires = DateTime.Now.AddMinutes(20);


                // Add it to the cookies collection
                Response.Cookies.Add(hc);

                // Redirect the user to the page they requested
                string strReturnURL = Request.Params["ReturnUrl"].ToString();
                if(strReturnURL != "") Response.Redirect(strReturnURL);
            }
            else
            {
                // Show a message that the credentials are invalid
                lblInvalidPassword.Visible = false;
            }
        }
    End Sub
    }
}
```

This code relies on three tables in Microsoft SQL Server to store the credentials: Users, Roles, and UserRoleMappings. Figure 7.2 shows the relationships between these tables. Listing 7.18 is a script that can be used to create the tables and stored procedures that are used by the login.aspx page.

This document is created with the unregistered version of CHM2PDF Pilot

Figure 7.2. The relationships between the Users, Roles, and UserRoleMappings tables.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Impersonation

Impersonation is the capability for the ASP.NET process to run in the context of a particular user identity. It is said to "impersonate" the logged-in user and is capable of using the logged-in user's identity to act on the user's behalf.

By default, Impersonation is not enabled in ASP.NET. You might initially doubt this because if you look at the User.Identity.Name property with Windows authentication enabled, it will be set to the logged-in user. This, however, is not the identity that the ASP.NET application is using to access resources. The identity that is shown here is used during URL authorization and file authorization, but will not be used as the identity when calling other base class functionality.

Determining Identity with WindowsIdentity

To determine the identity under which the ASP.NET process is running, you can use a static method, GetCurrent, of the WindowsIdentity to return the identity under which the current process is executing. When you examine the name returned by GetCurrent, you will see that you are really running under the SYSTEM account.

Listings 7.20 and 7.21 show a page that compares the results from User.Identity and WindowsIdentity.GetCurrent().

Listing 7.20 Webform1.aspx

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false"
Inherits="DefaultImpersonation.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
intellisense/ie5">
    </HEAD>
    <body>
        <form id="Form1" method="post" runat="server">
            <P>
                <STRONG style="FONT-FAMILY: Verdana">Windows Impersonation</STRONG>
            </P>
            <TABLE style="FONT-FAMILY: Verdana" cellSpacing="1" cellPadding="1"
width="300" border="1">
                <TR>
                    <TD colspan="2">
                        <STRONG>User</STRONG>
                    </TD>
                </TR>
                <TR>
                    <TD>
                        IsAuthenticated:
                    </TD>
                    <TD>
                        <asp:Label id="lblUserIsAuthenticated"
runat="server"></asp:Label>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
                    </TD>
                </TR>
                <TR>
                    <TD>
                        Authentication Type:
                    </TD>
                    <TD>
                        <asp:Label id="lblUserAuthenticationType"
runat="server"></asp:Label>
                    </TD>
                </TR>
                <TR>
                    <TD>
                        Name:
                    </TD>
                    <TD>
                        <asp:Label id="lblUserName" runat="server"></asp:Label>
                    </TD>
                </TR>
                <TR>
                    <TD>
                    </TD>
                    <TD>
                    </TD>
                </TR>
                <TR>
                    <TD colspan="2">
                        <STRONG>WindowsIdentity</STRONG>
                    </TD>
                </TR>
                <TR>
                    <TD>
                        IsAuthenticated
                    </TD>
                    <TD>
                        <asp:Label id="lblWIIsAuthenticated"
runat="server"></asp:Label>
                    </TD>
                </TR>
                <TR>
                    <TD>
                        AuthenticationType
                    </TD>
                    <TD>
                        <asp:Label id="lblWIAuthenticationType"
runat="server"></asp:Label>
                    </TD>
                </TR>
                <TR>
                    <TD>
                        Name:
                    </TD>
                    <TD>
                        <asp:Label id="lblWIName" runat="server"></asp:Label>
                    </TD>
                </TR>
                <TR>
                    <TD>
                    </TD>
                    <TD>
                    </TD>
                </TR>
            </TABLE>
        </form>
    </body>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
</HTML>
```

Listing 7.21 The Code Behind Class for WebForm1.aspx Showing the Difference Between User.Identity and WindowsIdentity.GetCurrent() with Impersonation Disabled

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace DefaultImpersonation
{
    /// <summary>
    /// Summary description for WebForm1.
    /// </summary>
    public class WebForm1 : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.Label lblUserIsAuthenticated;
        protected System.Web.UI.WebControls.Label lblUserAuthenticationType;
        protected System.Web.UI.WebControls.Label lblUserName;
        protected System.Web.UI.WebControls.Label lblWIIsAuthenticated;
        protected System.Web.UI.WebControls.Label lblWIAuthenticationType;
        protected System.Web.UI.WebControls.Label lblWIName;

        public WebForm1()
        {
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
            this.lblUserIsAuthenticated.Text =
User.Identity.IsAuthenticated.ToString();
            this.lblUserAuthenticationType.Text = User.Identity.
AuthenticationType.ToString();
            this.lblUserName.Text = User.Identity.Name;

            System.Security.Principal.WindowsIdentity wi = System.Security.
Principal.WindowsIdentity.GetCurrent();
            this.lblWIAuthenticationType.Text = wi.AuthenticationType.ToString();
            this.lblWIIsAuthenticated.Text = wi.IsAuthenticated.ToString();
            this.lblWIName.Text = wi.Name;
        }

        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
        }

        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        {
            this.Load += new System.EventHandler(this.Page_Load);

        }
        #endregion
    }
}
```

Running this page returns the following:

User	
IsAuthenticated:	True
Authentication Type:	NTLM
Name:	KINSMAN\ckinsman
WindowsIdentity	
IsAuthenticated:	True
AuthenticationType:	NTLM
Name:	NT AUTHORITY\SYSTEM

Now, turn impersonation on. This is done by adding the <identity> element to the web.config and setting the impersonate attribute to true. The modified web.config is shown in Listing 7.22.

Listing 7.22 Web.config Set Up for Impersonation

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <identity impersonate="true" />
        <compilation defaultLanguage="c#" debug="true" />
        <authentication mode="Windows" />
        <authorization>
            <allow users="*" /> <!-- Allow all users -->
        </authorization>
    </system.web>
</configuration>
```

If you run the same page shown in Listing 7.21, you will find that the names returned by WindowsIdentity.GetCurrent and by User.Identity are the same. The output from the page will look like this:

User	
IsAuthenticated:	True
Authentication Type:	NTLM
Name:	KINSMAN\ckinsman
WindowsIdentity	
IsAuthenticated:	True
AuthenticationType:	NTLM
Name:	KINSMAN\ckinsman

This document is created with the unregistered version of CHM2PDF Pilot

Now calls to access resources will operate as though you were the logged-in user, in my case, KINSMAN\ckinsman. What happens with an unauthenticated or anonymous user when impersonation is enabled? In this case, the ASP.NET behavior closely mirrors ASP.old behavior. The user appears to be unauthenticated and IsAuthenticated returns false. If you check WindowsIdentity.IsAuthenticated, however, you will see that it has IsAuthenticated set to true. It has to be, because this is the identity that ASP.NET is running under. It is impossible for the process to be running with no associated identity. So what identity is it using in this case? It is using the IUSR_<machinename> identity that IIS creates during installation. The same page run in a directory with anonymous enabled will look like this:

User	
IsAuthenticated:	False
Authentication Type:	
Name:	
WindowsIdentity	
IsAuthenticated:	True
AuthenticationType:	NTLM
Name:	KINSMAN\IUSR_STEPTOE

The machine I ran this code on was named STEPTOE, and as you would expect, the identity that the ASP.NET process is running under is shown to be IUSR_STEPTOE.

Hard-Coded Impersonation

ASP.NET provides an additional option for impersonation. It is possible within the <identity> element to hard-code the account that you would like ASP.NET to run under. If you hard-code the account into the identity element, this account is used to run the ASP.NET process regardless of whether the user is authenticated or not. Let's take a look at this. Listing 7.23 shows the modified web.config with a hard-coded user account.

Listing 7.23 Web.config with a Hard-Coded Impersonation Account

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <identity impersonate="true" userName="KINSMAN\ChrisKinsman"
             password="password" />
        <compilation defaultLanguage="c#" debug="true" />
        <authentication mode="Windows" />
        <authorization>
            <allow users="*" /> <!-- Allow all users -->
        </authorization>
    </system.web>
</configuration>
```

If you modify the web.config as shown in Listing 7.23 and run the same page with anonymous enabled, you will get the following output:

User	
IsAuthenticated:	False

This document is created with the unregistered version of CHM2PDF Pilot

Authentication Type:	
Name:	
WindowsIdentity	
IsAuthenticated:	True
AuthenticationType:	NTLM
Name:	KINSMAN\ChrisKinsman

Notice that the name is now set to the account that you specified in web.config. If you disable anonymous, you will get this output:

User	
IsAuthenticated:	True
Authentication Type:	NTLM
Name:	KINSMAN\ckinsman
WindowsIdentity	
IsAuthenticated:	True
AuthenticationType:	NTLM
Name:	KINSMAN\ChrisKinsman

Notice in this case that the name differs. Both are domain accounts. The WindowsIdentity.GetCurrent().Name is the account that I hard-coded in the web.config. The User.Identity.Name is the account that I used to authenticate the Web request.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 8. HttpHandlers and HttpModules

IN THIS CHAPTER

- An Overview of ASP.NET Request Handling
- HttpModules
- HttpHandlers
- Dynamic Handler Assignment
- Class Reference

In ASP.old, you just couldn't do certain things using ASP. If you wanted to create something similar to the output caching in ASP.NET, you were forced to step outside ASP.old and use Internet Server API (ISAPI) filters. If you wanted to create a program that handled all files with a certain extension, you had to step outside ASP. If you wanted to write something that participated in the processing of each and every page, you had to step outside ASP.

One of the goals of ASP.NET was to allow you to do everything you could potentially conceive of related to Web programming directly in the product. It shouldn't limit you. To that end, Microsoft added two new concepts: HttpModules and HttpHandlers. These did not exist inside of ASP.old. To find analogous functionality, you had to step outside into the world of ISAPI programming. HttpModules and HttpHandlers are fairly similar to ISAPI filters, but they implement slightly different functionality.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

An Overview of ASP.NET Request Handling

To understand how HttpModules and HttpHandlers fit into the scheme of things, you have to understand the way that ASP.NET handles a request. When a request is received by Internet Information Server (IIS), it looks at the extension to determine which ISAPI filter should handle the request. For any of the supported file extensions, such as .aspx or .asmx, the answer is aspnet_isapi.dll. When ASP.NET fires up, it performs almost the same process again. It looks at the request and compares it to the <httpHandlers> section of the .config file. By default, machine.config maps .aspx files to the PageHandlerFactory and .asmx files to the WebServiceHandlerFactory. This mapping determines the HttpHandler (class) that is responsible for handling the request. With the concept of mapping, you can create a new HttpHandler and map it to a new type of request. In fact, this is exactly what Microsoft did with Trace.axd. You will find that it is a new HttpHandler that is registered in machine.config for any request path that ends in trace.axd.

While processing a request received from IIS, ASP.NET raises several events. They are raised in the following order:

1.

1. BeginRequest
1.
2.

2. AuthenticateRequest
2.
3.

3. AuthorizeRequest
3.
4.

4. AcquireRequestState
4.
5.

5. ResolveRequestCache
5.
6.

6. Page Constructor
6.
7.

7. PreRequestHandlerExecute
7.
8.

8. Page.Init
8.
9.

This document is created with the unregistered version of CHM2PDF Pilot

9. Page.Load
9.
10.

10. PostRequestHandlerExecute
10.
11.

11. ReleaseRequestState
11.
12.

12. UpdateRequestCache
12.
13.

13. EndRequest
13.
14.

14. PreSendRequestHeaders
14.
15.

15. PreSendRequestContent
15.

The items in bold represent several of the page-level events that are raised during the execution of a page. Each of these events can be sunk, providing opportunities to participate in the processing of each page in an application. In Chapter 7, "Security," we looked at handling the AuthenticateRequest and AuthorizeRequest events.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

HttpModules

Many of these events can be sunk in the Global.asax of an application. By doing this, however, you limit the functionality to that application. To sink these events in a more reusable fashion, create an HttpModule. By adding a single line to the machine.config, your HttpModule affects all applications on the machine; by adding instead a single line to the web.config file, your HttpModule affects just that one application. The line to load an HttpModule looks like the following:

```
        <httpModules>
            <add type="SimpleModules.SimpleHttpModule, SimpleModules"
name="SimpleHttpModule" />
        </httpModules>
```

Let's take a look at a couple of sample HttpModules that handle some of the events on this class.

A Simple BeginRequest and EndRequest Module

BeginRequest is the first event to fire when processing a request. EndRequest is almost the last event to fire. Let's write an HttpModule that sinks these events and uses them to time stamp the output HTML with the time that the request began processing and when it finished processing. This information might be useful if you were trying to profile a group of pages.

We will create this as our first HttpModule. First, we need to create a class. This class will implement the IHttpModule interface. To implement this interface, we need to supply two members: Init and Dispose. When ASP.NET loads our HttpModule to participate in processing a request, a reference to the HttpApplication object is passed to the Init method. We will then save a reference to this in a member variable for use later in our module. We will also wire up several event handlers off the HttpApplication object.

After we have implemented IHttpModule, we can get into doing the things that are specific to our task. In this example, we need to create event handlers for BeginRequest and EndRequest. We do this by first creating our functions like this:

```
public void BeginRequest(object sender, EventArgs e)
```

Next we need to wire them up in the Init method that is part of the IhttpModule interface like this:

```
application.BeginRequest += new System.EventHandler(BeginRequest);
```

Inside of BeginRequest and EndRequest, we will utilize a saved reference for HttpApplication to write into the output stream a comment tag containing the date and time. The complete HttpModule is shown in Listing 8.1.

Listing 8.1 Implementation of a Module That Stamps the Begin and End Times into the Page

```
using System;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
using System.Web;

namespace SimpleModules
{
    /// <summary>
    /// Summary description for BeginEnd.
    /// <add type="SimpleModules.BeginEnd, SimpleModules" name="BeginEnd" />
    /// </summary>
    public class BeginEnd : IHttpModule
    {
        private HttpApplication mApplication;

        public void Init(System.Web.HttpApplication application)
        {
            // Wire up beginrequest
            application.BeginRequest += new System.EventHandler(BeginRequest);
            // Wire up endrequest
            application.EndRequest += new System.EventHandler(EndRequest);
            // Save the application
            mApplication = application;
        }

        public void BeginRequest(object sender, EventArgs e)
        {
            mApplication.Response.Write("<!-- Begin Request Time: " +
DateTime.Now.ToString("HH:mm:ss.fffffff") + " -->");
        }

        public void EndRequest(object sender, EventArgs e)
        {
            mApplication.Response.Write("<!-- End Request Time: " +
DateTime.Now.ToString("HH:mm:ss.fffffff") + " -->");
        }

        public void Dispose()
        {
        }

    }
}
```

To get this module to execute for a single application, we need to place it into the /bin directory and modify the web.config to include it in the httpModules section. The web.config should look like Listing 8.2.

Listing 8.2 The web.config to Load the BeginEnd HttpModule

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <httpModules>
            <add type="SimpleModules.BeginEnd, SimpleModules" name="BeginEnd" />
        </httpModules>
    </system.web>
</configuration>
```

Now if we fire off a page in this application root, we will see the time stamps introduced as comments into the HTML. A sample page output is shown in Listing 8.3.

Listing 8.3 The View Source of a Page That Has Been Affected by the BeginEnd Module

This document is created with the unregistered version of CHM2PDF Pilot

```
<!-- Begin Request Time: 19:02:04.1024016 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
    <head>
        <title>Begin End</title>
    </head>
    <body MS_POSITIONING="GridLayout">
        <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value="dDwxNDEzNDIyOTIxOzs+" />

            Time:
            8/23/2001 7:02:04 PM
        </form>
    </body>
</html>
<!-- End Request Time: 19:02:04.4729344 -->
```

Right now, the module works with a single application. Move it to machine.config and register the assembly in the global assembly cache, and every ASP.NET page on the entire server would suddenly get these time stamps! This is clearly an incredibly powerful technique.

Filtering Output

The preceding example showed how to insert content into the output using Response.Write(). What if you want to filter the content in the page? Perhaps you are writing an advertising system that needs to be able to find certain tags in a page and replace them with an advertisement. Although this is a common type of task, this task is a bit tougher to do. No property on the response object allows you to retrieve the contents of a page and modify it in situ. If you think about how ASP.NET sends pages to the client, however, you can understand why this is so. Depending on the buffering state and the programmer's use of Response.Flush(), the entire page may never exist on the server. Instead, it may be streamed to the client in dribs and drabs. However, ASP.NET by default enables buffering, so it certainly would have been nice to give us access to that buffer. Perhaps in v.Next (the next version) the object model will be updated to allow this access.

So how do you get the page output? As it turns out, you don't get it—you filter it. It is possible to put a filter in place that inserts itself between ASP.NET and the client. As ASP.NET streams data back to the user, your "filter" can alter it. This filtering is done using the base classes in the .NET framework. .NET provides an abstract class called a Stream. The Stream class is used as a pattern for writing to memory, files, and even sockets. It should come as no surprise then that ASP.NET gives you access to the stream that is connected to the client via the Response.Filter property.

To filter the page output, create an object that derives from Stream and pass it the Response.Filter property. Then, set the Response.Filter property to this object. Now when ASP.NET sends page output to the client, it is actually sending the page output to your object. You can then modify the content as you see fit, and when done with it, you write it to the client stream that was passed to your constructor.

This is easier to show than describe, so let's take a look at some code. Listing 8.4 shows the complete source for the ad insertion filter AdInserter.cs. Like the previous example, we implement IHttpModule. The difference is that in the BeginRequest event handler, we create an instance of the AdStream object, passing it the Response.Filter, which contains a stream pointed at the user. We then take the stream object and set the Response.Filter property to it.

This document is created with the unregistered version of CHM2PDF Pilot

Now the interesting work is actually done in AdStream. This is our "filter." Everything up to the Write() method toward the bottom is just implementing the required stream members. The Write() method is where things get interesting. ASP.NET calls Write() when it wants to send data to the client. The data is passed into Write() as a byte array. Byte arrays are great if we want to inspect things character by character, but in this case we are more interested in dealing in strings so that we can do some pattern matching. To convert the byte array to a string, use the UTF8Encoding class. This class converts a byte array to a Unicode string using UTF8 encoding. The result string is placed into a StringBuilder so that we can do simple replacement operations on it.

Strings are immutable, so simple string concatenations behind the scenes are really creating and destroying the underlying strings, causing a performance drain. The StringBuilder is a much more efficient way to do operations on a string. In this case, we are looking for the <adinsert></adinsert> tags, but this is a simplified task just for this example. In real life, you should instead search for <adinsert> only, do a string scan to find the </adinsert>, and then—based on position—replace what is between them. For simplicity, here we are replacing the exact match in this sample with a string that's derived by taking a random entry from the astrAds array. In a real ad insertion engine, this step would also be more complicated, most likely entailing a selection algorithm against a cache of items from a backing database store. Finally, the resulting string is written to the client stream using a StreamWriter, which supports writing a string to a stream without first having to convert it to a byte array.

Listing 8.4 A Simple Ad Insertion Engine That Replaces <adinsert> Tags with an Ad

```
    public class AdInserter : System.Web.IHttpModule
    {
        private System.Web.HttpApplication mApplication;

        public AdInserter()
        {
        }

        public void Init(System.Web.HttpApplication application)
        {
            // Wire up beginrequest
            application.BeginRequest += new System.EventHandler(BeginRequest);
            // Save the application
            mApplication = application;
        }

        public void BeginRequest(Object sender, EventArgs e)
        {
            // Create a new filter
            AdStream mStreamFilter = new AdStream(mApplication.Response.Filter);
            // Insert it onto the page
            mApplication.Response.Filter = mStreamFilter;
        }

        // AdStream filter
        public class AdStream : System.IO.Stream
        {
            // The ads to insert
            string[] astrAds = new string[] {"<adinsert><img
src=\"deep_468x60a.gif\"></adinsert>", "<adinsert><img
src=\"deepASPNET_color.gif\"></adinsert>"};

            //    The stream to the client
            private System.IO.Stream moStream;
            // Used to track properties not supported by the client stream
            private long mlLength;
            private long mlPosition;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
// An easy way to write a stream to the client
private System.IO.StreamWriter mSR;

public AdStream(System.IO.Stream stream)
{
    // Save of the stream back to the client
    moStream = stream;
    // Create a streamwriter for later use
    mSR = new System.IO.StreamWriter(moStream);
}

public override bool CanRead
{
    get
    {
        return false;
    }
}

public override bool CanSeek
{
    get
    {
        return true;
    }
}

public override bool CanWrite
{
    get
    {
        return true;
    }
}

public override long Length
{
    get
    {
        return mlLength;
    }
}

public override long Position
{
    get
    {
        return mlPosition;
    }
    set
    {
        mlPosition = value;
    }
}

public override int Read(Byte[] buffer, int offset, int count)
{
    throw new NotSupportedException();
}

public override long Seek(long offset, System.IO.SeekOrigin direction)
{
    return moStream.Seek(offset, direction);
}
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            public override void SetLength(long length)
            {
                mlLength = length;
            }

            public override void Close()
            {
                moStream.Close();
            }

            public override void Flush()
            {
                moStream.Flush();
            }

            public override void Write(byte[] buffer, int offset, int count)
            {
                System.Text.UTF8Encoding utf8 = new System.Text.UTF8Encoding();
                // Get the string into a stringbuilder
                System.Text.StringBuilder strBuff = new
System.Text.StringBuilder(utf8.GetString(buffer));
                // Random number used to grab ads
                Random oRandom = new Random(DateTime.Now.Millisecond);
                int iRandom = oRandom.Next(astrAds.GetLowerBound(0),
astrAds.GetUpperBound(0));
                // Go through and find the <adinsert></adinsert> tags
                strBuff.Replace("<adinsert></adinsert>", astrAds[iRandom]);
                // Write to the stream
                mSR.Write(strBuff.ToString());
            }
        }

        public void Dispose()
        {
        }
    }
```

The end result is a page that contains text from one of the elements in astrAds. Listing 8.5 shows the resulting HTML.

Listing 8.5 The Output from AdInserter.cs to a Page with <adinsert> Tags

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
    <head>
        <title></title>
        <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
                    content="http://schemas.microsoft.com/intellisense/ie5">
    </head>
    <body MS_POSITIONING="GridLayout">
        <adinsert><img src="deep_468x60a.gif"></adinsert><br>
        <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value="dDwtMTI3OTMzNDM4NDs7Pg==" />

            Time:
            8/23/2001 8:27:12 PM</form>
    </body>
</html>
```

This document is created with the unregistered version of CHM2PDF Pilot

Note the <img> tag that was inserted by the filter between the <adinsert> tags.

Forking the Filter

Filters work great if the task at hand calls for modifying the content as it streams to the client. Some tasks, however, don't fit this model. Suppose that you want to create something similar to the OutputCache in ASP.NET. For this to work, you need to have the entire contents of the page available after it has been written to the client. You might be thinking, "No problem, the stream has a read method." As it turns out, HttpResponseStream, which is the stream that ASP.NET uses to respond to a request, doesn't support the read operation. If you attempt to use it, you will get an UnsupportedException thrown. To make this idea work, your stream implementation must "Fork" the data written to it. One copy will be written to the client stream. The other copy will be written to an in-memory buffer that can then be read from at a later time. This way, when request processing is over, we can still access the content of the page.

The next example implements a very simplistic caching mechanism. It has an internal hash table that it uses to store pages that are keyed on the request URL. This example also uses two new events: ResolveRequestCache and UpdateRequestCache. You may wonder why two new events are needed. ResolveRequestCache is the appropriate event in this case because BeginRequest happens prior to the authentication and authorization stages. If you checked the cache for a page before those events fired, you could potentially return a cached page to an unauthorized user. That clearly would be undesirable. UpdateRequestCache is to place an executed page into the cache when it is done executing. Listing 8.6 contains the implementation of SimpleCache.

Listing 8.6 Implementation of SimpleCache, an Output-Caching Mechanism

```
using System;
using System.Web;
using System.Collections;

namespace SimpleModules
{
    /// <summary>
    /// Summary description for SimpleCache.
    /// </summary>
    public class SimpleCache : IHttpModule
    {
        // The stored application
        private HttpApplication mApplication;
        // Hash to store cached pages
        Hashtable mHash = new Hashtable();

        public void Init(HttpApplication app)
        {
            // Store off the application object
            mApplication = app;
            // Wire up our event handlers
            mApplication.ResolveRequestCache += new
System.EventHandler(this.ResolveRequestCache);
            mApplication.UpdateRequestCache += new
System.EventHandler(this.UpdateRequestCache);
        }

        public void Dispose()
        {
        }

        private void ResolveRequestCache(object sender, System.EventArgs e)
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        {
            // is the url in the cache?
            if(mHash.Contains(mApplication.Request.Url))
            {
                // Write it back from the cache
                mApplication.Response.Write(mHash[mApplication.Request.Url].
ToString());
                // Finish the request
                mApplication.CompleteRequest();
            }
            else
            {
                // Create a new filter
                CacheStream mStreamFilter = new
CacheStream(mApplication.Response.Filter);
                // Insert it onto the page
                mApplication.Response.Filter = mStreamFilter;
                // Save a reference to the filter in the request context so we can
grab it in UpdateRequestCache
                mApplication.Context.Items.Add("mStreamFilter", mStreamFilter);
            }
        }

        private void UpdateRequestCache(object sender, System.EventArgs e)
        {
            if(!mHash.Contains(mApplication.Request.Url))
            {
                // Grab the CacheStream out of the context
                CacheStream mStreamFilter = (CacheStream)
mApplication.Context.Items["mStreamFilter"];
                // Remove the reference to the filter
                mApplication.Context.Items.Remove("mStreamFilter");
                // Create a buffer
                byte[] bBuffer = new byte[mStreamFilter.Length];
                // Rewind the stream
                mStreamFilter.Position = 0;
                // Get the bytes
                mStreamFilter.Read(bBuffer, 0, (int)mStreamFilter.Length);
                // Convert to a string
                System.Text.UTF8Encoding utf8 = new System.Text.UTF8Encoding();
                System.Text.StringBuilder strBuff = new
System.Text.StringBuilder(utf8.GetString(bBuffer));
                // Insert the cached timestamp
                strBuff.Insert(0, "<!-- Cached: " + DateTime.Now.ToString("r") + "
-->");
                // Save it away
                mHash.Add(mApplication.Request.Url, strBuff.ToString());
            }
        }

        public class CacheStream : System.IO.Stream
        {
            private System.IO.MemoryStream moMemoryStream =
new System.IO.MemoryStream();
            private System.IO.Stream moStream;

            public CacheStream(System.IO.Stream stream)
            {
                moStream = stream;
            }

            public override bool CanRead
            {
                get
                {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            return true;
        }
    }

    public override bool CanWrite
    {
        get
        {
            return true;
        }
    }

    public override bool CanSeek
    {
        get
        {
            return true;
        }
    }

    public override long Length
    {
        get
        {
            return moMemoryStream.Length;
        }
    }

    public override long Position
    {
        get
        {
            return moMemoryStream.Position;
        }
        set
        {
            moMemoryStream.Position = value;
        }
    }

    public override int Read(byte[] buffer, int offset, int count)
    {
        return moMemoryStream.Read(buffer, offset, count);
    }

    public override long Seek(long offset, System.IO.SeekOrigin direction)
    {
        return moMemoryStream.Seek(offset, direction);
    }

    public override void SetLength(long length)
    {
        moMemoryStream.SetLength(length);
    }

    public override void Close()
    {
        moStream.Close();
    }

    public override void Flush()
    {
        moStream.Flush();
    }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            public override void Write(byte[] buffer, int offset, int count)
            {
                moStream.Write(buffer, offset, count);
                moMemoryStream.Write(buffer, offset, count);
            }
        }
    }
}
```

The pattern should be familiar by now. First, implement IHttpModule and save off a copy of the application. ResolveRequestCache is where things start to diverge from prior examples. In ResolveRequestCache, look in mHash to see if a cached copy of the page already exists. Call the Contains method, passing the URL of the request to determine if it is in the cache. If it is, retrieve the string from mHash, Response.Write it to the client, and then call HttpApplication.CompleteRequest. This call short-circuits execution of the request and causes ASP.NET to bypass the rest of the steps and stream the result back to the client. If the page is not in the cache, place an instance of CacheStream into Response.Filter, and also place a reference to CacheStream into HttpContext.Items. This reference is needed because the Response.Filter property always returns the stream that points to the client, even after it's set to point to a different stream. That way, multiple filters can be inserted and each can act on the stream. In this case, however, we need to get access to CacheStream later on during the UpdateRequestCache event.

To facilitate communication between events in HttpModules and/or HttpModules themselves, the HttpContext provides the items collection that allows data to be associated with the request. In this case, use it to store a reference to CacheStream. CacheStream inherits Stream and acts as the forking filter. Everything that is written to CacheStream is also written to an internal MemoryStream. CacheStream, unlike the previous examples, supports the Read method. When Read is called, information from the internal MemoryStream is returned. When UpdateRequestCache finally fires, it checks again to see if the current request is already in mHash. If it isn't, grab the CacheStream from the HttpContext and retrieve the copy of the page data that it contains. Add a comment to the beginning of the page data that stamps it with the date and time that the page was cached. This page is then placed into mHash, keyed off the URL. That's it! The OutputCacheModule in real life, of course, does considerably more than this, including aging of items from the cache and varying by parameters, but this HttpModule effectively demonstrates how to use the Filter property to get at the content of the page.

An Error Module

One of the coolest new application events in ASP.NET is the Error event. As mentioned before, with an HttpModule you can sink this event in an application-specific way in Global.asax. You can redirect the user away from the error page to some other part of the site that is more appropriate than just an error message. It might be interesting to sink the error event in a module, however, to provide a non–application-specific piece of functionality. A common idea is to log the error to the event log for later analysis or perhaps even to e-mail it to the Webmaster. This can be done in an application-independent way, which indicates the need for an HttpModule.

Listing 8.7 shows an HttpModule that logs the error information to an event log and e-mails the Webmaster with the error information. It first attempts to connect to an event log called "ASP.NET ErrorModule," which is created if it doesn't already exist. Next, it gathers the error information from the HttpApplication.Context.Error property. This property returns the exception that was thrown during the processing of this request. Several of the Exception properties are bundled into a string, which is then logged to the event log. Finally, the error is sent to the Webmaster using the SmtpMailClass.

Listing 8.7 An HttpModule That Handles Errors in an Application by Writing Them to the Event Log and

This document is created with the unregistered version of CHM2PDF Pilot

E-mailing the Webmaster

```
public class ErrorModule : IHttpModule
{
    private const string strEVENTLOGNAME = "ASP.NET ErrorModule";

    private HttpApplication mApplication;

    public void Init(HttpApplication application)
    {
        // Save off the application
        mApplication = application;

        // Wire up the error handler
        mApplication.Error += new System.EventHandler(this.ErrorHandler);
    }

    private void ErrorHandler(object sender, EventArgs e)
    {
        // Create the event source if it doesn't exist
        if(!EventLog.SourceExists(strEVENTLOGNAME))
            EventLog.CreateEventSource(strEVENTLOGNAME, strEVENTLOGNAME + " Log");

        // Create an event log instance and point it at the event source
        EventLog el = new EventLog();
        el.Source = strEVENTLOGNAME;

        // Create the error text
        string strErrorMessage = "An uncaught exception was thrown in your
application\r\nUrl: " + mApplication.Request.Url.ToString() + "\r\nMessage:" +
mApplication.Context.Error.Message + "\r\nStack Trace:" +
mApplication.Context.Error.StackTrace;

        // Write the event log entry
        el.WriteEntry(strErrorMessage, EventLogEntryType.Error);

        // Mail the message to the web master
        System.Web.Mail.SmtpMail.Send("webserver@vergentsoftware.com",
"ckinsman@vergentsoftware.com", "Web Site Error", strErrorMessage);
    }

    public void Dispose()
    {
    }
}
```

This code results in the event log entry shown in Figure 8.1 and the e-mail message shown in Figure 8.2.

Figure 8.1. The resulting event log entry.

This document is created with the unregistered version of CHM2PDF Pilot

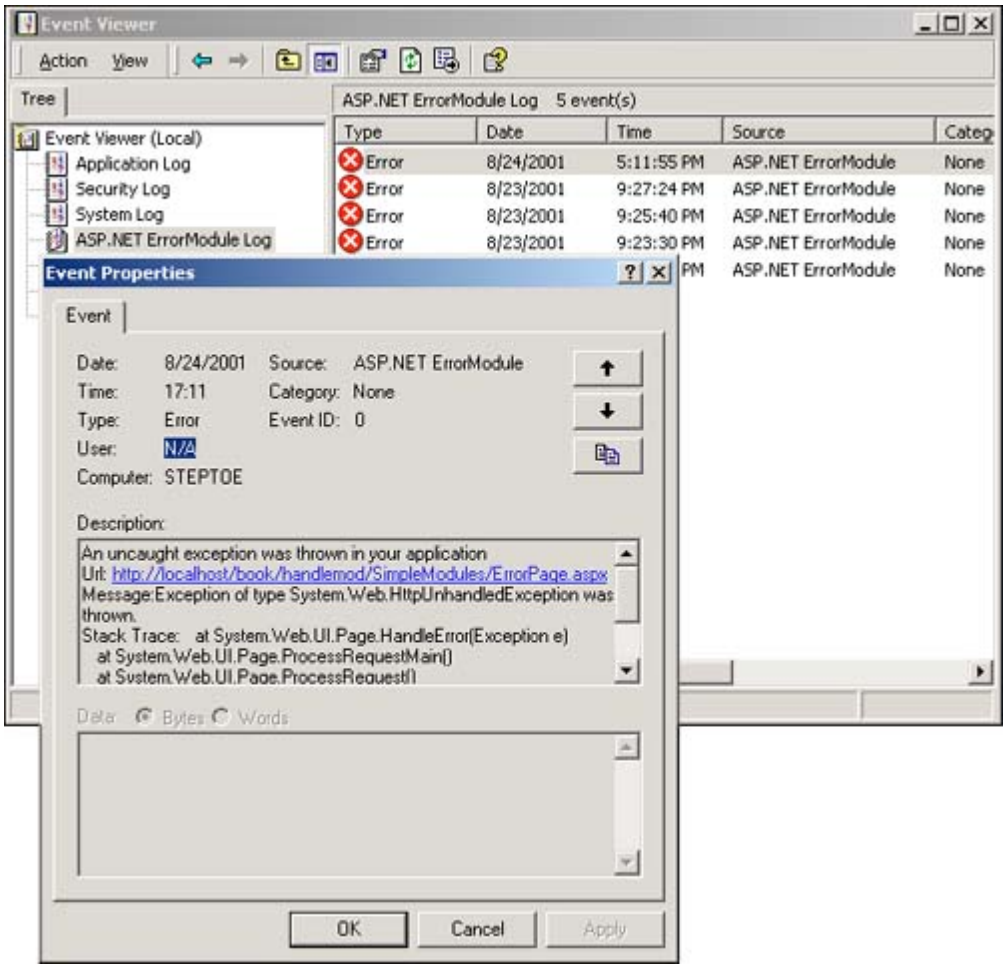
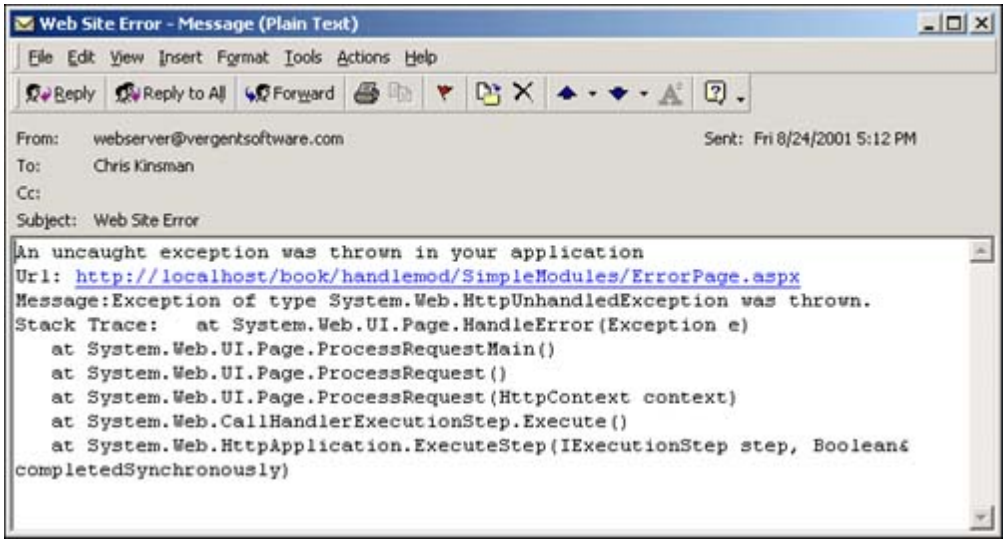


Figure 8.2. The resulting e-mail.



Notice that this HttpModule doesn't actually do any redirection. That is expected to be handled in an application-specific way. The sample shows another event handler defined in Global.asax for the Error event. This event handler is responsible for redirecting the user to a friendly error page. Both error event handlers fire during the processing of the request. This fact is important because it points out that multiple HttpModules can each be handling the same events. Listing 8.8 shows the global.asax.

Listing 8.8 Global.asax Does the Actual Redirection in an Application-Specific Way

```
public class Global : System.Web.HttpApplication
{
    private void InitializeComponent()
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    {
        this.Error += new System.EventHandler(this.Application_Error);
    }


    protected void Application_Error(object sender, EventArgs e)
    {
        Response.Redirect("friendlyerror.htm");
    }
}
```

Typically you wouldn't just redirect to another page in the event handler. You would normally do something along the lines of logging the error or notifying the administrator. If you just want to show another page instead of the error, also check out the <CustomErrors> element of the web.config, which is discussed in Chapter 5, "Configuration and Deployment."

Raising Events from an HttpModule

As mentioned previously, HttpModules are intended to be generic, containing no application logic. In many cases, however, you may want the developer of the application to tie application-specific code to your HttpModule. One way to do this is to raise events as part of your processing that the developer can sink in Global.asax to provide application-specific processing. Several of the built-in HttpModules raise events of this nature.

The way this is done is a little odd. No explicit event wireup is done. Instead, events are wired based on a naming convention. If you have a public event delegate in your code of the form:

```
public event EventHandler MyEvent
```

You can then put an event handler in the global.asax in the form friendlymodulename_eventname. When ASP.NET loads your HttpModule, it will dynamically wire up the event in the module to the event handler in the global.asax for you, based on the matching signatures. Listing 8.9 shows an HttpModule that raises an event in the global.asax of the application. It defines MyEvent and then raises it as part of the processing of BeginRequest.

Listing 8.9 An HttpHandler That Raises an Event in Global.asax

```
public class EventRaise : IHttpModule
{
    private HttpApplication mApplication;

    public event EventHandler MyEvent;

    public void Init(HttpApplication application)
    {
        // Save off the application object
        mApplication = application;

        // Wire up begin request
        mApplication.BeginRequest += new System.EventHandler(this.BeginRequest);
    }
    private void BeginRequest(object sender, EventArgs e)
    {
        OnMyEvent(new EventArgs());
    }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    private void OnMyEvent(EventArgs e)
    {
        if(MyEvent!=null)
            MyEvent(this, e);
    }

    public void Dispose()
    {
    }
}
```

Listing 8.10 shows the global.asax handling the event, based on the friendly name of the HttpModule, EventRaise, and the name of the event, OnMyEvent.

Listing 8.10 The global.asax That Sinks the OnMyEvent Event from the HttpModule

```
public class Global : System.Web.HttpApplication
{
    protected void EventRaise_MyEvent(object sender, EventArgs e)
    {
        Response.Write("MyEventFired!");
    }
}
```

Authentication Modules

In the previous chapter, we wrote an AuthenticateRequest handler that was used to do role mapping based on Forms authentication with a custom ticket. None of the code in AuthenticateRequest was really application specific. It could easily be abstracted into an HttpModule that sinks the AuthenticateRequest event and can then be reused in many other applications. Converting this code to work in an HttpModule is straightforward. Listing 8.11 shows AuthModule, an implementation of the functionality from the DbFormUrl Listing 7.16 in Chapter 7.

Listing 8.11 AuthenticateRequest in a Web Module for the DbFormUrl Example in Chapter 7

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Web;
using System.Web.SessionState;

namespace DBFormURL
{
    /// <summary>
    /// Summary description for Global.
    /// </summary>
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(Object sender, EventArgs e)
        {
            Application["DSN"] = "SERVER=localhost;UID=sa;PWD=;DATABASE=
SecuritySample";
        }

        protected void Application_AuthenticateRequest(object sender, EventArgs e)
        {
            // Make sure the user has been authenticated
            // This event fires for unauthenticated users also
            if(Request.IsAuthenticated)
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            {
                // Get the users identity
                System.Web.Security.FormsIdentity fiUser  =
(System.Web.Security.FormsIdentity)User.Identity;
                // Get the ticket
                System.Web.Security.FormsAuthenticationTicket at = fiUser.Ticket;
                // Grab out the roles
                string strRoles = at.UserData;
                // Renew the ticket if need be
                System.Web.Security.FormsAuthenticationTicket ticket =
System.Web.Security.FormsAuthentication.RenewTicketIfOld(at);
                if(ticket!=at)
                {
                    // Get the encrypted version of the ticket
                    string strEncrypted =
System.Web.Security.FormsAuthentication.Encrypt(ticket);

                    // Put it into a cookie
                    HttpCookie hc = new HttpCookie(System.Web.Security.
FormsAuthentication.FormsCookieName, strEncrypted);
                    hc.Expires = DateTime.Now.AddMinutes(20);


                    // Add it to the cookies collection
                    Response.Cookies.Add(hc);
                }

                // Create a new principal which includes our role information from
the cookie
                HttpContext.Current.User = new System.Security.Principal.
GenericPrincipal(fiUser, strRoles.Split(','));
            }
        }
    }
}
```

Rewriting Paths

Occasionally, a technique that you come up with for conveying information in a URL doesn't fit the standard model for URLs. A great example of this is the cookieless session management that we looked at in Chapter 4, "State Management and Caching." The URLs used in cookieless session management take on the following form:

http://localhost/sessionid/default.aspx

Where the sessionid part varies on a user-by-user basis. This is an invalid URL in the normal context of ASP.NET, so how is it handled? Behind the scenes, the cookieless Session state HttpModule uses a method of the HttpApplication, RewritePath(). RewritePath() allows you to take an incoming URL and change it to point to a different page. This is not a redirect, which requires a round trip to the client. It is also not a Server.Transfer; it happens prior to the PageHandlerFactory executing any code in a page.

RewritePath() allows the cookieless Session state HttpModule to change the preceding URL that ASP.NET looks for to the following:

http://localhost/default.aspx

This document is created with the unregistered version of CHM2PDF Pilot

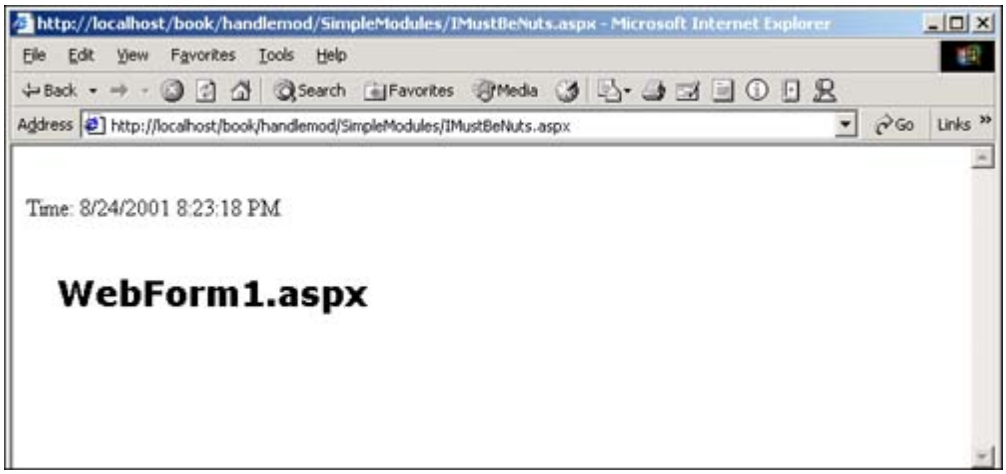
The URL in the user's browser remains unchanged—there's no noticeable difference. Let's take a look at an HttpModule that does something of this sort. The RewritePath module in Listing 8.12 sinks the BeginRequest event. Inside this event, it rewrites any request that is received by ASP.NET to instead point to the webform1.aspx file that is in the application root.

Listing 8.12 RewritePath Module That Changes Every Request to Map to webform1.aspx

```
public class RewritePath : IHttpModule
{
    private HttpApplication mApplication;

    public void Init(HttpApplication application)
    {
        // Save off the application
        mApplication = application;

        // Wire up the begin request
        mApplication.BeginRequest += new EventHandler(this.RewritePathHandler);
    }

    private void RewritePathHandler(object sender, EventArgs e)
    {
        mApplication.Context.RewritePath(mApplication.Request.ApplicationPath +
"/webform1.aspx");
    }

    public void Dispose()
    {
    }
}
```

It doesn't matter what you type in as a URL, because as long as it ends in .aspx, you will see the content of webform1.aspx in the browser, even though the typed-in URL is persisted in the address bar of your browser. Figure 8.3 shows an attempt to browse to a fictitious URL and the resulting page that shows WebForm1.aspx.

Figure 8.3. The result of typing in a fictitious URL with the RewritePath module in place.



for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

HttpHandlers

Whereas HttpModules are designed to participate in the processing of a request, HttpHandlers are designed to be the endpoint for the processing of a request. As mentioned earlier, an HttpHandler provides a way to define new page processors that handle new types of programming models. Table 8.1 shows the HttpHandlers provided by ASP.NET.

Table 8.1. Built In ASP.NET HttpHandlers

HttpHandler	Purpose
PageHandlerFactory	Processes .aspx pages.
WebServiceHandlerFactory	Processes .asmx XML Web services.
HttpForbiddenHandler	Yields an error message indicating that a type of page is not in service. By default, all .asax, .vb, .cs, .ascx, .config, .csproj, .vbproj, and .webinfo files are mapped to this in machine.config.
StaticFileHandler	Delivers any page that isn't specifically mapped, such as .html, .htm, and .jpg.
TraceHandler	Shows the page containing all of the trace output.

ASP.NET provides different handlers for ASP.NET pages and Web services. Each knows how to handle the files associated with the extension appropriately. HttpHandlers don't have to be backed by files, however. ASP.NET allows the HttpHandler to provide the entire response to a request. Listing 8.13 shows a very simple HttpHandler that displays a Hello World type of page.

Listing 8.13 Hello World from an HttpHandler

```
using System.Web;

namespace Handlers
    {
    public class SimpleHandler : IHttpHandler
        {

        public void ProcessRequest(HttpContext context)
            {
            context.Response.Write("<HTML><BODY>");
            context.Response.Write("Hello from SimpleHandler");
            context.Response.Write("</BODY></HTML>");
            }

        public bool IsReusable
            {
            get
                {
                return true;
                }
            }
        }
    }
```

This document is created with the unregistered version of CHM2PDF Pilot

This code implements the IHttpHandler interface, which describes only one method and one property. The IsReusable property lets ASP.NET know if it can reuse an instance of an HttpHandler or if it needs to re-create it from scratch each time. The ProcessRequest() method is called to do the work of the HttpHandler. In our simple handler, we output a very trivial HTML page that writes a message to the browser.

To get this simple handler to work, you need to do two things. First, you need to add a new application mapping to map an extension to ASP.NET. This mapping is required to make sure that IIS calls ASP.NET when it receives a request for a page with that extension. Without the mapping, ASP.NET is never invoked, and unless the HttpHandler happens to have a matching page, you will receive a 404 Not Found error.

To add a new application mapping, perform these steps in the Internet Services Manager.

1.

1. Select the application root of your application in Internet Services Manager.
1.
2.

2. Open the Property page.
2.
3.

3. Select the Directory tab.
3.
4.

4. Click the Configuration button.
4.
5.

5. Select the App Mappings tab.
5.
6.

6. Click the Add button and create an application mapping to aspnet_isapi.dll for the extension you are interested in. For this example, define the extension as .hello.
6.

The next step is to modify the web.config or machine.config files to map the extension to the class you have created. The <httpHandlers> section of web.config defines the handlers that ASP.NET will load. By adding a single line like the following:

```
<add verb="GET" path="*.hello" type="handlers.SimpleHandler, handlers" />
```

ASP.NET will now call your HttpHandler whenever a page with the extension .hello is called. Note the verb attribute. This indicates for which HTTP/1.1 verbs the action will be performed. Valid verbs include GET, PUT, and POST. If you want to handle any type of request for that URL, you can use a wildcard of *.

This document is created with the unregistered version of CHM2PDF Pilot

After all these steps are complete, whenever the user types a URL ending in .hello at any path within the application root, SimpleHandler will get called. If no files exist in the application root, all of the following URLs are valid:

-
- http://localhost/book/handlemod/handlers/junk/asdfa/asdfas/WebForm1.hello
-
-
- http://localhost/book/handlemod/handlers/WebForm1.hello
-
-
- http://localhost/book/handlemod/handlers/.hello
-

The resulting output is the very simple HTML page provided by the handler.

Dynamic Reporting

The next sample is going to do something a little more involved, combining SQL and XML. SQL Server 2000 is able to output data as XML and XSL, making a powerful combination. Let's write an HttpHandler that handles a new file extension, .xsql. In this case there will actually be physical .xsql files on disk. These files are just the XSL templates that should be applied to the XML output from SQL Server. Our handler will expect each request to these files to include a SQL parameter. This parameter indicates the query that should be run and then merged with the XSL template. This combination allows us to run any SQL that can be output as XML and dynamically bind it to an XSL template. It's a pretty powerful concept.

Let's take a look at the code in the HttpHandler first. Listing 8.14 shows the SqlHandler.

Listing 8.14 SqlHandler Transforms XML SQL Queries from SQL Server with XSL Templates

```
using System;
using System.Data.SqlClient;

namespace Handlers
{
    /// <summary>
    /// Summary description for SqlHandler.
    /// </summary>
    public class SqlHandler : System.Web.IHttpHandler
    {
        public SqlHandler()
        {
        }

        // Call like this
        // http://localhost/csharp/handlers/tableviewer.xsql=select * from authors
for xml auto, elements
        public void ProcessRequest(System.Web.HttpContext context)
        {
            System.IO.FileStream fs = null;
            SqlConnection cn = null;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            try
            {
                // Get the sql
                string strSql = context.Request["SQL"];

                // Setup a DB connection
                cn = new
SqlConnection("SERVER=localhost;UID=sa;PWD=;DATABASE=pubs;");
                // Open the connection
                cn.Open();
                // Create a command
                SqlCommand cmd = new SqlCommand(strSql, cn);
                // Get a data reader reference
                SqlDataReader dr;
                // Execute the sql
                dr = cmd.ExecuteReader();

                // Get a buffer
                System.Text.StringBuilder strBuff = new System.Text.
StringBuilder("<?xml version=\"1.0\" encoding=\"utf-8\" ?>\r\n");
                // Encapsulate with root element
                strBuff.Append("<results>");
                // Get all the rows
                while(dr.Read())
                {
                    strBuff.Append(dr.GetString(0));
                }
                // Add the ending element
                strBuff.Append("</results>\r\n");
                // Close the connection
                cn.Close();
                // Load XML into document
                System.Xml.XmlDocument xd = new System.Xml.XmlDocument();
                // Load it up
                xd.LoadXml(strBuff.ToString());

                // Attempt to open the xslt
                fs = new System.IO.FileStream(context.Request.PhysicalPath,
System.IO.FileMode.Open);
                // Load it into a navigator
                System.Xml.XPath.XPathDocument xpn = new System.Xml.XPath.
XPathDocument(new System.Xml.XmlTextReader(fs));

                // Close the file
                fs.Close();

                // Create a transform
                System.Xml.Xsl.XslTransform xslt = new System.Xml.Xsl.
XslTransform();
                // Load it
                xslt.Load(xpn);

                // Transform it to the output stream
                xslt.Transform(xd.CreateNavigator(), null, context.Response.
Output);
            }
            catch(Exception e)
            {
                // Write an error message
                context.Response.Write("<body><html>Invalid xsql query<Br>Message: "
+ e.Message + "</html></body>");
            }
            finally
            {
                // Close the file stream
```

This document is created with the unregistered version of CHM2PDF Pilot

```
                if(fs!=null) fs.Close();
                // Close the db connection
                if(cn!=null) cn.Close();
            }

        }

        public bool IsReusable
        {
            get
            {
                return true;
            }
        }

    }
}
```

Right at the beginning, wrap this code in a try/catch block. If any part of the code fails, it will write an error message back to the user indicating that a problem occurred with the query and including the exception text.

Next, grab the SQL parameter from the request object and use it to run a query against SQL Server. The HttpHandler expects that the SQL query includes the "for xml auto, elements" clause. Next, the HttpHandler retrieves all the data from the query using a DataReader. The HttpHandler prepends and appends a root element because SQL Server does not create a unique root element by default. The HttpHandler reads only the first column because the "for xml" clause causes SQL Server 2000 to return a 1 column by N row result, in which each row contains a portion of the string 2033 characters or less in length. So, concatenate these fragments using a StringBuilder. Again, because strings are immutable, concatenation operations are expensive, requiring copying the string around in memory. For something like this where there could be a large number of concatenations, a StringBuilder makes more sense—it's very efficient to append text using a StringBuilder.

After all the data is loaded into the StringBuilder, go ahead and load it into an XMLDocument to prepare it for transforming. The next step is to load the actual XSL template. This was the URL that caused the HttpHandler to fire in the first place. Get its physical path using the PhysicalPath property of the request object, and then open it with a FileStream and load it into an XpathDocument via a XMLTextReader. Finally, an XSLTransform object is created, the XSL is loaded into it, and the transformation is performed. Pass the Response.Output property to the Transform method. This is a stream, which is one of the possible outputs for the Transform method.

Now we also need an XSL template. Listing 8.15 shows a generic XSL template that takes some XML and formats it as a table.

Listing 8.15 The XSL Template Used to Format the Output

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:msxsl="urn:schemas-microsoft-com:xslt" version="1.0">
    <xsl:template match="/">
        <xsl:apply-templates select="/*" />
    </xsl:template>
    <xsl:template match="/*">
        <html><body>
        <Table WIDTH="100%" BORDER="1" topmargin="0" leftmargin="0"
                    cellpadding="0" cellspacing="0">
```

This document is created with the unregistered version of CHM2PDF Pilot

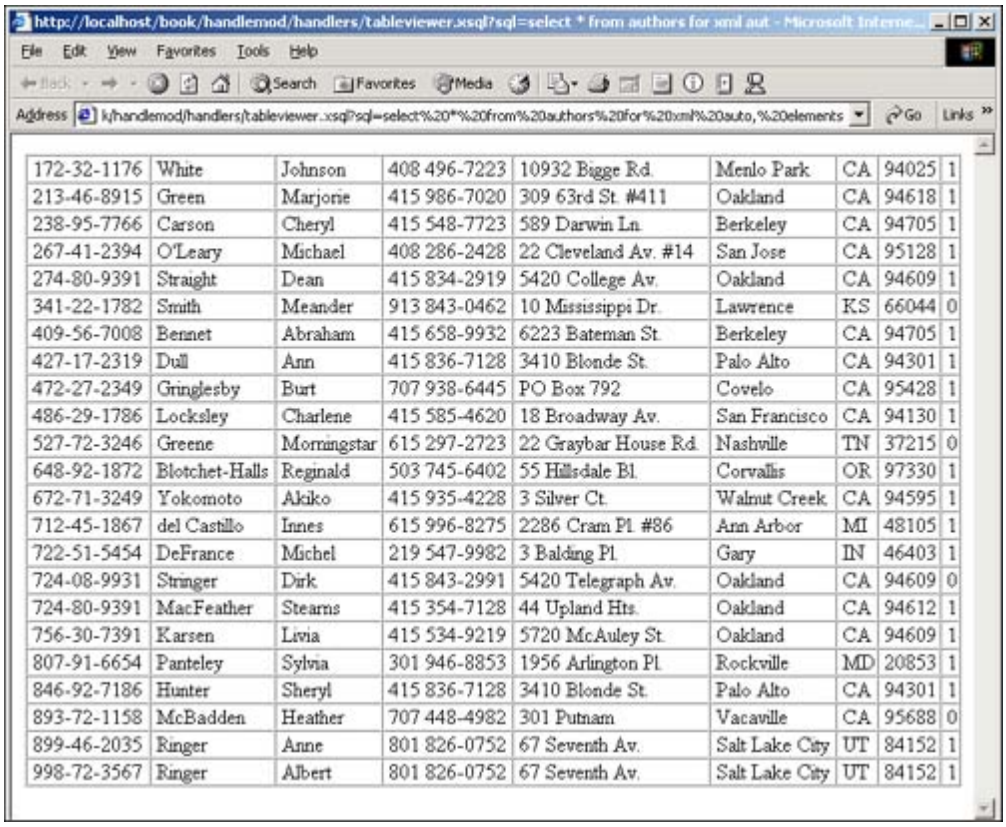
```
            <xsl:for-each select="./*">
                <tr>
                    <xsl:for-each select="./*">
                        <td> 
                                                    <xsl:value-of select="." />
                                                </td>
                    </xsl:for-each>
                </tr>
            </xsl:for-each>
        </Table>
        </body></html>
    </xsl:template>
</xsl:stylesheet>
```

This template is generic XSL that could be used for almost any query. Before we can test these two items (Listing 8.14 and Listing 8.15), we have to add the .xsql application mapping in Internet Service Manager. As noted previously, this routes requests for .xsql files to ASP.NET. We also need to add an entry in web.config to map requests for .xsql files to SqlHandler. Finally, we can run the code. We need to specify a URL of the following form:

```
http://localhost/book/handlemod/handlers/tableviewer.xsql?sql=select%20*%20from%20author
s%20for%20xml%20auto,%20elements
```

This includes the path to our XSL template as well as the URL-encoded SQL that we want it to run. The query contained in the preceding URL yields the output shown in Figure 8.4.

Figure 8.4. The output from our SqlHandler.


http://localhost/book/handlemod/handlers/tableviewer.xsql?sql=select * from authors for xml aut - Microsoft Interne...
File Edit View Favorites Tools Help
Search Favorites Media
Address k/handlemod/handlers/tableviewer.xsql?sql=select%20*%20from%20authors%20for%20xml%20auto,%20elements
Go Links

172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	1
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	1
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	1
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	1
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	1
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	66044	0
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	1
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	1
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428	1
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA	94130	1
527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	37215	0
648-92-1872	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	97330	1
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	94595	1
712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	48105	1
722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	46403	1
724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av.	Oakland	CA	94609	0
724-80-9391	MacFeather	Stearns	415 354-7128	44 Upland Hts.	Oakland	CA	94612	1
756-30-7391	Karsen	Livia	415 534-9219	5720 McAuley St.	Oakland	CA	94609	1
807-91-6654	Panteley	Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD	20853	1
846-92-7186	Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	1
893-72-1158	McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA	95688	0
899-46-2035	Ringer	Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT	84152	1
998-72-3567	Ringer	Albert	801 826-0752	67 Seventh Av.	Salt Lake City	UT	84152	1

NOTE

This document is created with the unregistered version of CHM2PDF Pilot

The preceding example is not incredibly secure since you have given a user a way to execute any arbitrary SQL on your server. This includes extended stored procedures and DDL.

Page Counter Handler

Graphical page counters were all the rage early on during the evolution of the Web, but it wasn't easy to create the images dynamically in ASP until now. The next example is an HttpHandler that can be called inside an image tag in any page, like this:

```
<img src="PageCounter.cntr">
```

As long as the HttpHandler is mapped, the path is irrelevant. Upon execution, the HttpHandler looks at the referrer to determine what page it is being displayed in. The referrer is used as the key of an internal hash table that contains the current page view count. (If you were to move this structure into production, you would need a more durable storage location than just a private hash table.) After the page view has been looked up, it is time to go to work. We get a new 1-pixel bitmap just so we can get a graphics object. Because we are doing this code in an HttpHandler, there is no "paint" method that comes with a pre-created object for us. By creating a 1-pixel bitmap, we can then obtain a graphics object for the bitmap. Next, we create a font. In this case, we are using a Verdana 14-point font, but the font specifics could be passed on the command line to dynamically select a font.

Everything is now in place to measure the page count string. After we know the measurements, it is time to create a new bitmap of the appropriate size. The bitmap is cleared, anti-aliasing is turned on, and the string is drawn into the new bitmap. We convert the bitmap to a GIF using the Save method. The final step is to stream it to the Response.Output stream after setting the ContentType to image/gif. Listing 8.16 shows the HttpHandler.

Listing 8.16 A Page Counter HttpHandler That Dynamically Generates Page Count GIF Files

```
using System;
using System.Drawing;

namespace SimpleHandler
{
    /// <summary>
    /// Summary description for PageCounter.
    /// </summary>
    public class PageCounter : System.Web.IHttpHandler
    {
        // object to hold our counters
        private System.Collections.Hashtable hPageCounter =
new System.Collections.Hashtable();

        public PageCounter()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        public void ProcessRequest(System.Web.HttpContext context)
        {
            int iPageCount = 1;
            string strUrl = context.Request.UrlReferrer.ToString();
```

This document is created with the unregistered version of CHM2PDF Pilot

```
            string strPageCount;

            if(hPageCounter.Contains(strUrl))
            {
                // Get the page count and increment by 1
                iPageCount = (int) hPageCounter[strUrl] + 1;
                // Create a string of the page count
                strPageCount= iPageCount.ToString();
                // Update the page count
                hPageCounter[strUrl] = iPageCount;
            }
            else
            {
                // Init the page count
                hPageCounter.Add(strUrl, 1);
                // Set the page count to 1
                strPageCount = iPageCount.ToString();
            }

            // Create a new bitmap of minimum size
            Bitmap b = new Bitmap(1,1);
            // Get a graphics surface
            Graphics g = Graphics.FromImage(b);
            // Create a font
            Font f = new Font("Verdana", 14);
            // Measure the string so we know how wide to make it
            SizeF s = g.MeasureString(strPageCount, f);

            // Create the proper size bitmap
            b = new Bitmap((int)s.Width, (int)s.Height);
            // Get the graphics surface again
            g = Graphics.FromImage(b);
            // Clear the background to white
            g.Clear(System.Drawing.Color.White);
            // Indicate antialiased text
            g.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAlias;

            // Draw the page count on the bitmap
            g.DrawString(strPageCount, f, new
SolidBrush(System.Drawing.Color.Black), 0, 0);
            g.Flush();

            // Output the graphic
            context.Response.ContentType = "image/gif";
            b.Save(context.Response.OutputStream, System.Drawing.Imaging.
ImageFormat.Gif);
        }

        public bool IsReusable
        {
            get
            {
                return true;
            }
        }
    }
}
```

The end result is an image in the page containing the count of how many times the page has been viewed.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Dynamic Handler Assignment

In some cases, you may want to dynamically determine at runtime the appropriate HttpHandler to call for handling a particular request. .NET provides a Factory design pattern that allows you to create a Factory that is responsible for creating the appropriate HttpHandler to deal with the request. This gives you some additional flexibility in creating HttpHandlers. You could look inside an associated file to determine which handler should be called.

The Factory pattern also provides a way for you to potentially pre-create a number of handlers and hand them to ASP.NET when it requests one, without the overhead of creating one each and every time.

Let's look at an example. Listing 8.17 shows a class that implements IHttpHandlerFactory. This class looks for an argument passed as part of the URL. If the value of this argument is "Chris", the ChrisHandler is returned to ASP.NET to handle the request. If the value of the argument is "Jeffrey", the JeffreyHandler is returned.

Listing 8.17 A Sample HttpHandlerFactory That Returns Different Handlers Based on the Name Parameter

```
using System;
using System.Web;
using System.Web.UI;

namespace Handlers
{
    /// <summary>
    /// Summary description for HandlerFactory.
    /// </summary>
    public class HandlerFactory : IHttpHandlerFactory
    {
        public IHttpHandler GetHandler(HttpContext context, string requestType,
string url, string pathTranslated)
        {

            // Check the name property
            if(context.Request["Name"] == "Chris")
                // If it's Chris return chris
                return new ChrisHandler();
            else
                // Else return Jeff
                return new JeffHandler();
        }

        // required to implement the interface
        public void ReleaseHandler(IHttpHandler handler)
        {
        }
    }

    /// The ChrisHandler
    ///
    public class ChrisHandler : IHttpHandler
    {
        public void ProcessRequest(HttpContext context)
        {
            context.Response.Write("<html><body>Chris</body></html>");
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        }

        public bool IsReusable
        {
            get
            {
                return true;
            }
        }

    }

    /// The JeffHandler
    ///
    public class JeffHandler : IHttpHandler
    {
        public void ProcessRequest(HttpContext context)
        {
            context.Response.Write("<html><body>Jeff</body></html>");
        }

        public bool IsReusable
        {
            get
            {
                return true;
            }
        }

    }

}
```

The Chris and Jeffrey handlers just write a simple document with the name Jeffrey or Chris. The HttpHandlerFactory is hooked up in Web.Config the same way an ordinary HttpHandler is hooked up.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Class Reference

This section provides a quick interface reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book, so for the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

HttpApplication

Member of System.Web.

Assembly: System.Web.dll.

Represents the top-level class or root object for ASP.NET. Almost everything to do with processing a request in ASP.NET hangs off of this class. This class is responsible for raising events during request processing.

Properties		
Application	Context	Modules
Request	Response	Server
Session	Site	User
Methods		
AddOnAcquireRequest-StateAsync	AddOnAuthenticate-RequestAsync	AddOnAuthorizeRequest-Async
AddOnBeginRequestAsync	AddOnEndRequest-Async	AddOnPostRequestHandler-ExecuteAsync
AddOnPreRequestHandler-ExecuteAsync	AddOnRelease-RequestStateAsync	AddOnResolveRequest-CacheAsync
AddOnUpdateRequestCacheAsync	CompleteRequest	Dispose
Equals	GetHashCode	GetType
GetVaryByCustomString	Init	ToString
Events		
AcquireRequestState	AuthenticateRequest	AuthorizeRequest
BeginRequest	EndRequest	PostRequestHandler-Execute
PreRequestHandlerExecute	PreSendRequestContent	PreSendRequestHeaders
ReleaseRequestState	ResolveRequestCache	UpdateRequestCache

IHttpModule

This document is created with the unregistered version of CHM2PDF Pilot

Member of System.Web.

Assembly: System.Web.dll.

This is the abstract class that all HttpModules must inherit from for ASP.NET to properly load them.

Methods	
Init	Dispose

EventLog

Member of System.Diagnostics.

Assembly: System.dll.

This class provides everything needed to interact with the event logs.

Static Methods		
CreateEventSource	Delete	DeleteEventSource
Exists	GetEventLogs	LogNameFromSourceName
SourceExists	WriteEntry	
Properties		
Container	EnableRaisingEvents	Entries
Log	LogDisplayName	MachineName
Site	Source	SynchronizingObject
Methods		
BeginInit	Clear	Close
CreateObjRef	Dispose	EndInit
Equals	GetHashCode	GetLifetimeService
GetType	InitializeLifetimeService	ToString
WriteEntry		
Events		
Disposed	EntryWritten	

SmtpMail

Member of System.Web.Mail.

This document is created with the unregistered version of CHM2PDF Pilot

Assembly: System.Web.dll.

This class is a wrapper for CDONTS, the simple SMTP server that is optionally installed with Windows 2000 and Windows XP.

Static Properties
SmtpServer
Static Methods
Send

IHttpHandler

Member of System.Web.

Assembly: System.Web.dll.

This is the abstract class that all HttpHandlers must implement.

Properties
IsReusable
Methods
ProcessRequest

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 9. Building User Controls and Server Controls

IN THIS CHAPTER

- Working with User Controls in Web Forms Applications
- Creating Server Controls

Many server-scripting environments, including ASP.old, made it difficult to reuse the code that composes Web-based user interfaces. The most common option for creating reusable code is the server-side include (SSI). With SSIs, you create an external file containing HTML and/or server script that can be referenced, or included, from any other file in your Web application.

SSIs are adequate and are supported on many types of server environments (including both ASP.old and ASP.NET) across different platforms. But the server-side include is a harsh mistress, quick to anger. One problem is that the SSI doesn't provide any standard functionality, only a way for including code functionality from an external file. That means that the way you get access to code in a given SSI may be totally different from the way you access code in another. Also, code in SSIs typically is not accessible from any language (the language of the SSI must be the language of your main page), and making a change in an SSI that's referenced from multiple files frequently causes code in linked files to break.

This paucity of options for code reuse is in stark contrast to development environments such as Visual Basic and Java, which both have a rich library of user-interface controls that developers can access in the form of reusable objects. (In general, presenting code in the form of objects ensures that the code is consistent and easy to understand; it also keeps the developer who uses the code from having to know much of anything about how the code works internally.)

ASP.NET adds support for several new kinds of reusable user-interface objects (including server controls and HTML controls), all of which were introduced in earlier chapters. This chapter discusses how to create your own user-interface objects.

ASP.NET enables you to create two types of custom user-interface objects in Web forms programming:

- User control—This control is an evolution of the functionality traditionally provided by server-side includes. With user controls, you can easily create a bit of user-interface functionality that is based in easily maintained

This document is created with the unregistered version of CHM2PDF Pilot

script. But unlike SSIs, Web forms user controls are fully object oriented, supporting properties, methods, and events.

-
-

- Custom server control—This control is a type of .NET component that also provides Web forms user-interface functionality, but in a way that takes advantage of the full spectrum of programmability features available in the .NET component development model. These features include inheritance of .NET UI component base classes, a powerful complexity-management tactic that prevents you from having to reinvent the wheel when creating custom components.
-

We discuss how to create both types of controls in this chapter. Because user controls are far easier to develop and are probably more commonly encountered, we'll cover those first.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Working with User Controls in Web Forms Applications

When you're interested in reusing a piece of user-interface functionality in an ASP.NET application, but you don't want to expose yourself to the full brunt of a precompiled server control, you can create a user control. User controls are simple to build in ASP.NET script and don't require the precompilation that server controls do.

To create a user control, you typically start with a basic HTML representation of the user interface you want. This is a convenient feature of user controls. It means you can use whatever HTML editing control you're most comfortable with to do the lion's share of control development.

For example, suppose you're building an information portal application in which search functionality is a key feature. You want to have a Search dialog box on every page of your application. To implement this, you can create a Search dialog box user control. The HTML representation of the control is shown in Listing 9.1.

Listing 9.1 Initial HTML Representation of the Search Dialog Box User Control

```
<table width="250" border="0" cellpadding="3" cellspacing="0">
  <tr>
    <td bgcolor="#000066"><font color="#FFFFFF">Search</font></td>
  </tr>
  <tr>
  <td align="center" bgcolor="#CCCCCC" height='75px'>
    <table width="98%" border="0">
        <tr>
          <td>Text:</td>
          <td>
            <input type="text" name="textfield">
          </td>
          <td>
            <input type="submit" name="Submit" value="Go">
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

You should notice a few important things about the HTML representation of this user control. First, notice that the HTML header tags (things such as <HTML>, <HEAD>, and <BODY>) are missing. This is the case because this control is intended to be the component of an existing page, not a page that stands alone. You should not include these tags when building a user control.

Next, notice that although this HTML contains form elements such as a text box and button, it does not contain an actual <FORM> tag. This is because the containing page is where the form tag should reside; form tags don't belong in user controls themselves.

Finally, user controls are always saved using the .ascx file extension. Giving your controls an .ascx extension easily identifies the code as a user control and prevents the Web server from serving up the file. You can verify this by

This document is created with the unregistered version of CHM2PDF Pilot

attempting to navigate to a file with an .ascx extension—the server will refuse to send it directly in the browser. Rather than navigating to them directly, user controls will be rendered only in the context of a hosting page, as we'll demonstrate later in this chapter.

A Search dialog box such as this one could be useful in many Web applications. If you were to use this bit of HTML again and again in different places, you would want to have some high-level programmatic control over its behavior. For example, you would definitely want to be able to specify the text displayed in the title bar and the Search prompt, as well as specify other cosmetic aspects such as the background color of the title bar and body of the control.

You can provide programmers who use your control the capability to change these values programmatically by exposing them as user-control properties. We'll describe how that works in the next section.

In addition to saving the control as a file with an .ascx extension, your user control can contain an additional element—a Control directive. This directive is similar to the Page directive that appears at the start of ASP.NET pages, but it contains special settings that are relevant to user controls.

The Control directive supports a subset of the attributes supported by the Page directive (described in Chapter 2, "Page Framework"). The attributes supported by the Control directive are the following:

-
- AutoEventWireup
-
-
- ClassName
-
-
- CompilerOptions
-
-
- Debug
-
-
- Description
-
-
- EnableViewState
-
-
- Explicit
-
-

This document is created with the unregistered version of CHM2PDF Pilot

- Inherits
-
-

- Language
-
-

- Src
-
-

- Strict
-
-

- WarningLevel
-

You use these settings nearly identically to the way you use them for pages; see Chapter 2 for more information on what they do and how they work.

The Inherits and Src attributes refer to code-behind functionality for the control itself. This enables you to separate code functionality from a user control the same way you do for a page. Code behind is also discussed in Chapter 2.

You'll notice that the Trace attribute is missing from the list of attributes found on the list of Control directives. The Trace attribute is not a part of the Control directive because trace mode functionality is controlled by the entire page and cannot be limited to a specific user control. For more information on tracing, see Chapter 3, "Debugging ASP.NET Applications."

Adding Properties to a User Control

You can give developers the capability to programmatically change elements of your user control by exposing certain elements of the control as public properties. By doing this, you enable customization of the control at a high level, without forcing developers to manually hack the HTML that composes your control.

To do this, you replace hard-coded HTML elements in your control with references to public variables or (preferably) property procedures. Listing 9.2 shows an example of the Search dialog box created in the previous section, this time with a property procedure for a TitleBarText property.

Listing 9.2 Search Dialog Box with a TitleBarText Property

```
<SCRIPT runat='server'>
  private String strTitleBarText;

  public String TitleBarText
  {
    get {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
      return strTitleBarText;
    }

    set {
    strTitleBarText = value;
    }
  }

</SCRIPT>
<table width="250" border="0" cellpadding="3" cellspacing="0">
  <tr>
    <td bgcolor="#000066"><font color="#FFFFFF"><% =TitleBarText %></font></td>
  </tr>
  <tr>
    <td align="center" bgcolor="#CCCCCC" height='75px'>
      <table width="98%" border="0">
        <tr>
          <td>Text:</td>
          <td>
            <input type="text" name="textfield">
          </td>
          <td>
            <input type="submit" name="Submit" value="Go">
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

The three things that changed in this version of the control are the inclusion of a private variable, strTitleBarText, to store the value of the title bar text, a property accessor function called TitleBarText, and a render block (shaded) in the middle of the HTML representation of the control.

When you include this code in the control, the HTML representation of the control will be different, depending on how the programmer of the containing page chooses to set these properties. For an example of the finished product, see the next section, "Programming a User Control in a Page."

It's common for controls such as this Search control to incorporate other properties to alter the appearance and behavior of the control. Your controls will probably contain properties for foreground and background color, height and width, and so on. These properties are very easy to implement in user controls with the techniques demonstrated here.

Programming a User Control in a Page

To use a user control in an ASP.NET Web forms page, you must first include a Register directive at the top of the page and then create a tag that represents an instance of that control, as you do with the other intrinsic ASP.NET Web forms controls.

Listing 9.3 gives an example of a typical Register directive for the Search user control we created in the previous section.

This document is created with the unregistered version of CHM2PDF Pilot

Listing 9.3 Example of a Minimal Register Directive Referencing a User Control in a Page

```
<%@ Register TagPrefix="MyControl"
             TagName="Search"
             Src="search.ascx" %>
```

Three attributes are required in the Register directive. The TagPrefix attribute gives your control a unique namespace so that its name doesn't collide with other controls that may have similar names. This prefix comes in handy when you need to use two types of controls, written by different authors but both coincidentally named Search, on the same page. However, rather than using the generic tag prefix MyControl, you should use a tag prefix that identifies yourself or your company. That way, a namespace collision between your control and someone else's will be less likely.

The TagName attribute provides a name that identifies your control class. Note that this is different from the name of the instance of the control you use to manipulate the control programmatically.

Finally, the Src attribute indicates where the source file (.ascx) of your user control is stored. This file does not have to be in the same directory as the page; it can be in a subdirectory.

After you've registered your control for use in a page, you can create an instance of the control using the same tag-based syntax you use for any other kind of Web form control in ASP.NET. For example, if the Register directive for your Search control looks like the previous example (in Listing 9.3), a tag that would create an instance of the control on an ASP.NET Web form looks like this:

```
<MyControl:Search id='Search1' runat='server' />
```

As with all Web form controls, you should ensure that your control tags are placed inside an ASP.NET form (that is, a FORM tag that contains the runat="server" attribute). If you don't do this, your page won't work. Also, as with other types of Web forms controls, you can assign default properties to the control two ways: in the tag itself or in the Page_Load event procedure (or both).

For example, suppose you're using the Search user control in an application that enables users to search an online personnel database. In this case, you might want to set certain properties of the user control in code to reflect the purpose of the search. Listing 9.4 provides an example.

Listing 9.4 Example of a Custom User Control Utilized in an ASP.NET Web Form

```
<%@ PAGE language='C#' debug='true' trace='false' %>
<%@ REGISTER TagPrefix='MyControl' TagName='Search' Src='search.ascx' %>
<html>
  <head>
    <title>ASP.NET Page </title>
  </head>
  <script runat='server'>
    void Page_Load(Object Sender, EventArgs e)
    {
      Search1.TitleBarText = "Personnel Search";
    }
  </script>
  <body>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    <form runat='server'>
      <MyControl:Search id='Search1' runat='server' />
    </form>
  </body>
</html>
```

In this example, the TitleBarText property is set in the page's Page_Load event procedure. But it could just as well have been set in the declaration tag for the control itself, like so:

```
<MyControl:Search id='Search1'
                  TitleBarText='PersonnelSearch'
                  runat='server' />
```

There's really no difference between the two techniques. Which one you use depends on your preference and whether the property setting is known when you author the page. If the property setting isn't known (that is, the property is derived from a calculation), you'll need to assign it in code.

Adding Methods to Your User Control

You can add methods to your user control the same way you add properties to controls. To add a method, you simply create a public function or subroutine in the SCRIPT tag contained in your user control.

For example, suppose you want the page programmer to be able to show an Advanced Search dialog box. To enable this, you might provide a ShowAdvanced method in your Search control. Listing 9.5 shows an example of a new version of the Search control that provides this functionality.

Listing 9.5 Search Control with ShowAdvanced Method to Provide Extended Features

```
<SCRIPT runat='server'>
  private String strTitleBarText;

  public String TitleBarText
  {
    get
    {
      return strTitleBarText;
    }

    set
    {
      strTitleBarText = value;
    }
  }

  public void ShowAdvanced()
  {
    // Public (control method)
    Advanced.Visible = true;
  }

</SCRIPT>
<table width="250" border="0" cellpadding="3" cellspacing="0">
  <tr>
    <td bgcolor="#000066"><font color="#FFFFFF">
        <% =TitleBarText %>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
      </font>
    </td>
  </tr>
  <tr>
    <td align="center" bgcolor="#CCCCCC" height='75px'>
      <table width="98%" border="0">
        <tr>
          <td>Text:</td>
          <td>
            <input type="text" name="textfield">
          </td>
          <td align="center">
            <input type="submit" name="Submit" value="Go">
          </td>
        </tr>
      </table>
    </td>
  </tr>
  <tr id='Advanced' visible='False' runat='server'>
    <td align="center" bgcolor="#CCCCCC" height='40'>
      <asp:checkbox id="ShowContractors" value="1" runat="server" />
      Show contractors</td>
  </tr>
</table>
```

The Advanced Search functionality is provided through an HTML server control, a table row control called Advanced. The Visible property of this control is set to false by default, but calling the ShowAdvanced method of the control sets it to true. In addition to displaying the row, all the controls contained in the row are automatically displayed as well. You could include code in ShowAdvanced to do other interesting work, such as setting defaults for the Advanced Search controls and so forth.

To call this method, the page programmer calls the Show method in the hosting page:

```
<script runat='server'>
  void Page_Load(Object Sender, EventArgs e)
  {
     Search1.TitleBarText = "Personnel Search";
    Search1.ShowAdvanced();
  }
</script>
```

The simplicity of this code highlights an important benefit of user controls (and componentized code in general)—adding new functionality to a component does not typically require that programmers who use that component make significant changes in their code to adapt to the new functionality. If you don't want to use the Search control's new functionality, simply leave it out of your method call.

Handling Events from a User Control

Your user control can handle events that are generated by controls contained by the user control. For example, you may choose to encapsulate Search functionality within the Search control itself. To do this, you write a handler for the event that kicks off the search, inserting any code you want in the event handler.

This version of the Search control demonstrates how to handle internal system events. For this example, when the

This document is created with the unregistered version of CHM2PDF Pilot

user clicks the OK button, the button's Click event is handled by the Search_Click event procedure, which then kicks off the actual search. We haven't talked about doing database queries yet (we get to that in Chapter 11, "Creating Database Applications with ADO.NET"), so we'll stub out the "real" search results with a label control. Listing 9.6 provides an example of the Search control.

Listing 9.6 Version of the Search Control That Handles an Internal Event

```
<SCRIPT runat='server'>
  private String strTitleBarText;

  public String TitleBarText
  {
    get{
      return strTitleBarText;
      }
    set{
      strTitleBarText = value;
      }
  }

  void Search_Click(Object Sender, EventArgs e)
  {
    SearchResults.Text = "Search results for '" +
            SearchText.Text   +   "' go here.";
  }

</SCRIPT>
<table width="250" border="0" cellpadding="3" cellspacing="0">
  <tr>
    <td bgcolor="#000066"><font color="#FFFFFF">
        <% =TitleBarText %>
      </font>
    </td>
  </tr>
  <tr>
    <td align="center" bgcolor="#CCCCCC" height='75px'>
      <table width="98%" border="0">
        <tr>
          <td>Text:</td>
          <td>
            <asp:textbox id='SearchText' runat='server' />
          </td>
          <td align="center">
            <asp:button id="GoSearch" onClick='Search_Click' text="Go"
runat='server' />
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
<br>
<asp:label id='SearchResults' runat='server' />
```

Note that you don't have to handle the Click event of the button if you don't want to. Because the user control exists on a form in an ASP.NET page, the control will generate a round trip to the server when its Go button is pressed. You can then handle the Search functionality in the hosting page if you didn't already handle it inside the control. There's no right way to accomplish this division of labor; encapsulating the Search functionality in the control makes the control simpler to use, but less flexible, because the programmer of the hosting page would have to change the control if she wanted to change the Search functionality.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Creating Server Controls

In the previous sections you learned how to create user controls. User controls are a fast and easy way to create reusable user-interface elements in ASP.NET, and they serve as a more structured and richer alternative to server-side includes. However, user controls don't take full advantage of the .NET component model. As a result, they are somewhat limited in terms of functionality.

User controls are compiled automatically in the background, just as ASP.NET pages are. The first time your page is accessed in the browser, the page and any user controls on that page are compiled. After the first compilation, the page and user controls are recompiled only if you change their source code.

In contrast, server controls must be compiled and deployed to the Web server ahead of time. Although this may seem like a bit of a pain, you can actually automate much of the compilation process ahead of time using the command-line compiler and a Windows batch file. This is the tactic we'll use in this chapter.

NOTE

In Visual Studio .NET, when you create a new project and select Web Server Control, VS.NET creates an actual server control project for you (in contrast to a user control). To create a user control in the development environment, you first create an ASP.NET Web Application and then add a user-control file to the project.

Creating a Basic Server Control

You create the most basic kind of compiled server controls by following these steps:

1.

1. Create a class that inherits from System.Web.UI.Control.
1.
2.

2. Override the Render method of the inherited Control class.
2.
3.

3. Optionally, add any properties, methods, or events that are appropriate to your control's functionality.
3.
4.

4. Optionally, give your control the capability to store and retrieve state using the State object.
4.

This document is created with the unregistered version of CHM2PDF Pilot

As an example, we'll start with a control whose content is completely static— literally, a "Hello, world" example. From there, we'll build more sophisticated and useful controls that have properties, methods, and events you can set programmatically.

NOTE

The properties, methods, and events of the System.Web.UI.Control class are listed at the end of Chapter 2.

The first step is to create a class contained in a namespace that inherits System.Web.UI.Control and overrides the Control class's Render method. Listing 9.7 shows an example.

Listing 9.7 "Hello World" Server Control Example

```
using System.Web.UI;     // Contains HtmlTextWriter and Control classes

namespace MyExample
{
  public class HelloWorld : Control
  {
    protected override void Render(HtmlTextWriter Output)
    {
      Output.Write("<B>Hello world!</B>");
    }

  }
}
```

After writing the code, the next step is to compile this code into a .NET component DLL and deploy it into the Web application directory. You can compile this code with the command-line compiler. The command is

```
Csc  /t:library /out:helloctrl.dll  helloctrl.cs /r:System.dll /r:System.Web.dll
```

When you compile this control, you must reference the libraries System.dll and System.Web.dll because they contain .NET framework libraries used in your code. The System.Web.dll library is required by every server control because it contains the Control base class from which ASP.NET server controls inherit. This namespace also contains the HtmlTextWriter class that is passed in to the Render method; as you can see from the code listing, this class is used by your code as a way to send HTML data to the browser.

NOTE

For now, we've intentionally glossed over the details of how command-line compilation works in .NET. You can find more detailed information on how to use the command-line compiler to create your components in the next section.

Compiling this code produces a binary, helloctrl.dll, which should be copied to a \bin subdirectory under a Web

This document is created with the unregistered version of CHM2PDF Pilot

application directory to be accessible from an ASP.NET application in that directory.

The final step is to create an ASP.NET page that references this control and invokes it. Listing 9.8 shows a minimal example of such a page.

Listing 9.8 ASP.NET Page That References the Custom "Hello World" Server Control

```
<%@ PAGE language='C#' debug='true' trace='false' %>
<%@ REGISTER TagPrefix='Jeffrey' Namespace='MyExample' Assembly='helloctrl' %>
<HTML>
  <HEAD>
    <TITLE>Hello World Server Control </TITLE>
  </HEAD>
  <BODY>
    <FORM runat='server'>
      <Jeffrey:HelloWorld id='Hello1' runat='server' />
    </FORM>
  </BODY>
</HTML>
```

The Register directive is crucial here; it's what makes the page aware of the control contained in the external library. Like the user controls we discussed earlier in this chapter, the REGISTER tag contains a TagPrefix attribute; this attribute is an arbitrary text name you give to distinguish your controls from like-named controls on the page. As you can see in the code example, we used the TagPrefix "Jeffrey"; the same tag prefix is used when referencing the control on the page.

The Register directive for a precompiled server control also contains two attributes not found in user-control registrations. The Namespace attribute references the control's namespace; it must be the same as the namespace you declared when you constructed the server control.

Finally, the Assembly attribute points the page to the name of the assembly in which the component class resides. This should be the filename of the compiled DLL, without the .DLL filename extension. Remember that for the page to find the DLL, the DLL must be copied to a \bin subdirectory beneath the Web application directory.

Assuming everything is in place, when you navigate to this page in the browser, you should be able to see the "Hello World" text emitted by the overridden Render method of the HelloWorld control. You can, of course, replace this simple HTML with any text you want by altering the output of the Render function and then recompiling and redeploying the server control.

Before we go into more detail on the more sophisticated features of server controls, we'll take a brief detour and cover the general steps involved in creating and deploying .NET components. Because server controls are a type of .NET component, it's important to have a basic understanding of how components are built in .NET.

Compiling Your Control as a .NET Component

Because server controls are a type of .NET component, they must be compiled separately from the ASP.NET page that hosts them. You can do this in a number of ways. You may choose to build and compile your control in Visual

This document is created with the unregistered version of CHM2PDF Pilot

Studio .NET, which is certainly okay. However, we prefer to compile our .NET components manually, using the command-line compiler. Although this takes a bit longer to set up, you may find that it's faster in the long run. It certainly gives you more control over what's going on, at any rate, and will probably prove to be more enlightening in terms of how your source code gets turned into a .NET component.

You may have created COM components in previous versions of Visual Studio or Visual Basic. The theory behind components in .NET is similar, although the implementation is different (in our opinion, component-based development is easier in many ways in .NET than it was in COM). Any .NET code can be packaged as an independently compiled component as long as the code contains a namespace and at least one public class.

Creating .NET Components Using the Command-Line Compiler

The first step to compiling a component using the command line is to add the location of your compiler to your computer's PATH variable. In Windows 2000 you do this by opening the System control panel, clicking the Advanced tab, clicking the Environment Variables button, and altering the PATH variable located in the System variables panel. A number of semicolon-delimited paths should already be in the PATH variable; you'll need to add something like

```
%SystemRoot%\Microsoft.NET\Framework\v1.0.2914
```

to the end of whatever's there. Ensure that you separate the original path and the Microsoft .NET directory string with a semicolon. Note that the location and name of the .NET framework compilers will almost certainly be different on your machine, depending on which version and build of .NET you're using. If you're in doubt as to which PATH setting to append, use Windows Explorer to do a search for the file csc.exe and use whatever directory that file is located in. The objective here is simply to provide a way for you to get to csc.exe from a command line without having to type the path to it every time you compile.

NOTE

You should be able to find compilers for at least three languages in the .NET directory; vbc.exe is the Visual Basic compiler, csc.exe is the one for C#, and jsc.exe is used for applications created in JScript, Microsoft's implementation of JavaScript. You can also see a number of DLLs that compose the .NET framework in this directory—files such as System.dll, System.Web.dll, System.Data.dll, and so forth.

After you've appended the location of the .NET compilers to your PATH setting, you can open a command window by selecting Start, Run, and then typing cmd into the Run dialog box. After the command window appears, you can test to make sure everything works by typing csc /help | more into the command window. If everything worked correctly, the numerous parameters for the C# compiler will appear in the window.

To compile a class to a component using the command-line compiler, you must specify that the compiled output (the "target") should be a component rather than a conventional executable. When compiling with csc, you do this using the command-line switch /target:library. The output of a build that uses the switch /target:library is a .NET component contained in a familiar DLL file.

This document is created with the unregistered version of CHM2PDF Pilot

Listing 9.9 shows an example of one of the smallest possible chunks of code that can be compiled into a .NET component.

Listing 9.9 Example of a Namespace and Class That Can Be Compiled into a .NET Component

```
using System;
namespace HelloComponent
{

  public class Hello
  {

    public String SayHello()
    {
      return "Hello, world!!!";
    }

  }

}
```

Note that this component is contained by a namespace and contains a single public function, SayHello, which returns a hard-wired text string.

Assuming that this code is contained in a file called hello.cs, you could compile this class to a .NET component called hello.dll by using the following command line:

```
csc /target:library /out:hello.dll hello.cs
```

You can see that compiling components is fairly straightforward, as long as you keep things simple. The tricky part with command-line compilation of components has to do with ensuring that you've included references to all the external libraries used by your component. For example, if your code uses classes found in the System.Data namespace (including subnamespaces such as System.Data.SqlClient), the command you use to build your component must contain a reference to the component System.Data.dll, or the build will fail because the compiler won't be able to find the external libraries your code refers to.

Therefore, when compiling, you must include references to external libraries using the /r switch. Your command line can have as many /r switches as it needs. For example, if your component references the System.Data and System.Xml namespaces, the command line you use to compile your component might look like this:

```
csc /out:mycomp.dll /target:library mysrc.cs
/r:System.Data.dll /r:System.XML.dll
```

More options are available in the command-line compiler, but these are all you'll need to know to get your server controls to compile.

Sooner or later you'll notice that your command-line compilation statements will become long and unwieldy. This is particularly the case when your project comprises many source files or when your project contains more than one or two references to external libraries. In this case, you'll probably want to automate the build process. A common way

This document is created with the unregistered version of CHM2PDF Pilot

to accomplish this is by creating a batch file. Listing 9.10 shows an example of a batch file that compiles a .NET component project.

Listing 9.10 Batch File That Compiles and Deploys a .NET Component Project Automatically

```
Csc /out:mycomp.dll /t:library mysrc1.cs mysrc2.cs /r:System.Data.dll
copy mycomp.dll c:\inetpub\wwwroot\myapp\bin
pause
```

This set of commands, all contained within the text file build.bat, is all that's needed to create and deploy the fictitious mycomp.dll component. Note that two source files are in this project (mysrc1.cs and mysrc2.cs), and we're using the /t abbreviation instead of /target in this example. When the compilation is complete, the batch file copies the newly compiled DLL to c:\inetpub\wwwroot\myapp\bin.

Remember that batch files must end in a .bat or .cmd extension. You can use any text editor (such as the onerous Notepad or the extremely whizzy shareware TextPad) to create them.

Deploying a Component in ASP.NET

Deploying components is much easier in .NET than it was using COM. In .NET, you need only to deploy the component to a \bin directory located beneath your application directory. That's it! No REGSVR32, no shutting down and restarting IIS—none of that jive.

The batch file demonstrated in the previous section also copied the DLL to the Web application's \bin subdirectory after the compiler ran. Copying the file to a directory where it can be tested immediately after compilation is, of course, optional, but it's a handy trick that you'll probably want to use often.

No difference exists between a server control and any other kind of component in .NET, so the technique you use to develop, compile, and deploy server controls is essentially the same that you would use to create middle-tier business-logic components that have no representation in the user interface.

Creating Composite Controls

In the previous section, you saw a simple example of how to create a simple, HTML-based control by overriding the Render method of System.Web.UI.Control. This is a straightforward way to emit a chunk of static HTML.

A composite control, on the other hand, is a control that comprises other controls (either HTML controls or server controls). A Search control comprising a text label, a text box, and a button is an example of a composite control. This control would be similar to the Search control described in the section on user controls earlier in this chapter. We will create a new version of this control as a server control in our example.

Composite server controls are a type of ASP.NET server control. You create them essentially the same way as you create server controls, starting by creating a class that inherits from the System.Web.UI.Control class. The technique to create a composite control is slightly different from creating normal server controls, however. Instead of overriding

This document is created with the unregistered version of CHM2PDF Pilot

the Render method of System.Web.UI.Control, as you do with normal server controls, you instead override the CreateChildControls method of the Control class. In this method, you add child controls to your control in two ways: by adding instances of existing server control objects to the Controls collection contained by the control object (inherited from the Control base class) and by inserting literal HTML into the control by way of the LiteralControl object. The LiteralControl object is used by ASP.NET for HTML elements that don't require server processing, so make sure that you don't plan on programmatically accessing any of the HTML you create using a LiteralControl.

For example, to build a composite server control similar to the Search user control described earlier in this chapter, you need at least three elements: a text box, a button to submit the search, and a separator to go between the text box and the button. In the user control examples, we used a table for this; for this example, we'll use an HTML nonbreaking space.

Listing 9.11 shows an example.

Listing 9.11 Creating a Compositional Control by Overriding the CreateChildControls Method of the Control Object

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace MyServerControl
{

  public class Search : Control
  {

    protected override void CreateChildControls()
    {
      TextBox txt =  new TextBox();
      Button btn = new Button();
      btn.Text = "Search";
      this.Controls.Add(txt);
      this.Controls.Add(new LiteralControl(" "));
      this.Controls.Add(btn);
    }

  }

}
```

You can compile this code to a file called search.dll by using the following command in a command prompt window:

```
csc /t:library /out:search.dll search.cs  /r:System.dll /r:System.Web.dll
```

By navigating to the page shown in Listing 9.12, you can test how the control appears in the page.

Listing 9.12 Page Created to Host an Instance of the Composite Search Server Control

```
<%@ Page Language="C#"%>
<%@ Register TagPrefix="demo" Namespace="MyServerControl" Assembly="search" %>
<HTML>
  <HEAD>
    <TITLE>ASP.NET Page</TITLE>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  </HEAD>
  <BODY>
    <FORM runat='server' ID="Form1">
      <demo:Search id='Search1' runat='server' />
    </FORM>
  </BODY>
</HTML>
```

If you copy the file search.dll to the Web application's \bin directory and navigate to this page, you should be able to see an instance of the Search control on the page, with a text box and command button.

Notice that within the control code, you can access the properties and methods of the contained controls. In this example, we specified a default text property for the Command button (using the assignment statement btn.Text = "Search").

Adding properties and methods to this control is done in the same way you add properties and methods to any class. To do this, you create a public variable or property procedure, or (for methods) a public subroutine or function.

In this example, we'll provide access to the Text property of the Search control. In this version of the control, the public Text property of your Search control just provides access to the Text property contained in the child TextBox control. The TextBox control does the work of storing and retrieving the text. (In object-oriented programming, the term for handing off an operation to a related or contained control is delegation.)

Listing 9.13 shows an example of using delegation to store and retrieve the Text property of the Search control within the Text property of the child TextBox control.

Listing 9.13 Using Delegation to Provide Access to the Text Property of a Child Control

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace MyServerControl
{

  public class Search1:Control
  {
    private TextBox txt;
    private String Text
    {
      get
      {
        EnsureChildControls();
        return txt.Text;
      }
      set
      {
        EnsureChildControls();
        txt.Text = value;
      }

    }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    protected override void  CreateChildControls()
    {
      TextBox txt = new TextBox();
      Button btn = new Button();

      btn.Text = "Search";
      txt.Text = "Testing this.";

      this.Controls.Add(txt);
      this.Controls.Add(new LiteralControl(" "));
      this.Controls.Add(btn);
    }


  }

}
```

In this version of the Search control, the TextBox control is defined at the class level rather than inside the CreateChildControls subroutine, as in the previous example. This extends the child control's lifetime to that of the class so we have an opportunity to access its Text property after it's created. You can see in the Text property procedure that the Text property of the contained TextBox control can be stored and retrieved. But in each case, there's a call to a function called EnsureChildControls() in there. A call to this function (actually a method of the Control base class) is required whenever you reference a property of a child control in a composite server control. This must be done because the page is loaded asynchronously; without an explicit call to EnsureChildControls, it's possible that your code will attempt to access a property of a child control that has not yet been loaded by the server.

Delegating to child controls is useful when your control is a composite control comprising two or more types of controls or when the number of properties of a child control you want to expose is limited. But what happens when you want to create a specialized type of existing server control? You may want your control to have most of or all the properties and methods of the existing control, plus a few additional members (or overridden members). You certainly would not want to write custom accessors for the 70+ members of the base TextBox class. In this case, you'll instead want to use inheritance to create your server control, subclassing an existing control to use its functionality.

Subclassing Existing Server Controls

In the previous section you saw an example of a composite control built from several existing controls. But we identified a problem—if you want to expose a large number of properties of a child control, you have to write a large number of property accessor functions that enable the page to get to the properties of the contained control.

Inheritance provides an alternative to containing an instance of a control and delegating to its members using accessor functions. A control that inherits from an existing control is said to be subclassed from that control.

You can create customized subclassed versions of existing ASP.NET server controls using inheritance, without writing tons of code. Through inheritance, you can use the existing control as a base class, adding new members or overriding existing members to provide enhanced functionality.

The technique for doing this is similar to creating any server control. But instead of inheriting from

This document is created with the unregistered version of CHM2PDF Pilot

System.Web.UI.Control, you instead inherit from whichever control you're interested in subclassing. Because all server controls ultimately inherit from System.Web.UI.Control, your subclassed control still satisfies the requirement that all server controls inherit from the abstract Control class.

For example, suppose you want to provide a custom text box that provides a large number of default values. You don't want to have to code these defaults every time you use the control, and you know you're going to use this kind of control again and again in the construction of your site, so creating a custom control that inherits from the basic TextBox control makes sense.

To accomplish this, you create a class that inherits from the System.Web.UI.WebControls.TextBox control, overriding the property of the text box, supplying your own formatting defaults in the object constructor. Listing 9.14 shows an example.

Listing 9.14 CustomTextBox Control That Contains a Set of Custom Formatting Defaults

```
using System;
using System.Drawing;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace MyServerControl
{

  public class CustomText : TextBox
  {

    public CustomText()
    {

      this.BackColor = Color.CornflowerBlue;
      this.Text = "Can I get this one in cornflower blue?";
      this.Font.Name = "Verdana";
      this.Font.Size = FontUnit.XSmall;
      this.Width = Unit.Pixel(300);
    }
  }

}
```

You can compile this control using the command line:

```
csc /t:library /out:CustomText.dll CustomText.cs /r:System.dll
/r:System.Web.dll /r:System.Drawing.dll
```

You can see that this code imports a namespace we haven't used before; the System.Drawing namespace is used to set a background color for the control. As a result, the assembly System.Drawing.dll must also be included in your compilation command.

The customized default properties for the text box are set in the control's constructor; you can specify any properties you want by using MyBase to indicate that you want to change a property derived from the base TextBox class. (The quotations are all from the movie Fight Club; I got sick of using "Hello, world" all the time.)

This document is created with the unregistered version of CHM2PDF Pilot

Because it inherits from the standard Web forms TextBox control, the subclassed CustomText control has all the properties, methods, and events of the normal TextBox control. In addition to using inherited properties, you could override properties, as well.

To demonstrate how the inherited Text property works, you can deploy the compiled control assembly to the Web server's \bin directory and navigate to a page that instantiates the control. Listing 9.15 shows a page that puts the control through its paces.

Listing 9.15 ASP.NET Page That Uses an Instance of the Subclassed CustomText Control

```
<%@ Page language='C#' debug='true' trace='false' %>
<%@ Register TagPrefix="demo" Namespace="MyServerControl"
Assembly="CustomText" %>
<SCRIPT runat='server'>

  void Change_Click( Object Sender,EventArgs e)
  {
    CustomText1.Text = "...because waste is a thief.";
  }

</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>ASP.NET Page</TITLE>
  </HEAD>
  <BODY>
    <FORM runat='server' ID="Form1">
      <demo:CustomText id='CustomText1' runat='server' /><BR>
      <asp:button runat='server' text='The First Rule of Fight Club'
            id='Change' onClick='Change_Click' />
    </FORM>
  </BODY>
</HTML>
```

You should be able to see when this page loads that the text box is an unappealing cornflower blue color with the font and text properties set as specified in the control's constructor code. By clicking the button, the control's Text property is set to the value specified in the Change_Click event procedure.

Events in the Life of a Server Control

A number of events are fired by every ASP.NET server control. These events are primarily inherited from the server control's base class, System.Web.UI.Control; you typically handle these events to perform initialization tasks related to the control and the data it displays.

The sequence of events raised by a server control are listed in Table 9.1.

Table 9.1. Events Raised by System.Web.UI.Control in Custom Server Controls

Phase	Description	Found In
DataBinding	The control has been bound to a data source.	Control class

This document is created with the unregistered version of CHM2PDF Pilot

Init	Instructs the control to set its default property values. These values are retained for the life of the HTTP request.	Control class
LoadViewState	View state information is loaded into the appropriate properties of the control.	Control class
LoadPostData	Process information from the incoming form.	IPostBackDataHandler interface
Load	Triggered when the control is loaded into the Page object.	Control class
RaisePostDataChangedEvent	Informs the page that the control's state has changed.	IPostBackDataHandler interface
RaisePostBackEvent	Enables the control to process postback events.	IPostBackEventHandler interface
PreRender	Triggered just before the control is sent to the client.	Control class
Unload	The server control is unloaded from memory.	Control class

You will often find it useful to write code to respond to these events in your server control's lifetime. To handle these events, you override the corresponding "On" method (OnInit, OnLoad, OnDataBinding, and so forth) provided by the Control base class.

Binding Controls to Data

The ASP.NET controls you create can be bound to data. By enabling data binding, you can make it easy for application developers to use your control in database applications.

You provide support for data binding in your application by overriding the OnDataBinding method provided by System.Web.UI.Control, the base class of your control.

You set the data source consumed by your control by assigning a data object to the DataSource property of the control. If you're working with relational data, the data source can be a DataSet or DataReader object (introduced in Chapter 11); but you can also bind your control to a .NET array or collection object.

Generating Postback in Server Controls

As we discussed in Chapter 2, postback is the process of sending a form to the Web server for processing. When this happens, server script can process the contents of the form and perform useful tasks with the form data; this is the basis of Web applications.

A number of server controls provided with ASP.NET generate postback, including the Button and ImageButton controls, the LinkButton, HtmlButton, HtmlInputButton, HtmlImageButton, and DropDownList, CheckBoxList, or RadioButtonList (with AutoPostBack set to true). To generate a postback, your control must emit a chunk of

This document is created with the unregistered version of CHM2PDF Pilot

JavaScript code because the process of submitting a form to the server must always be initiated on the client side; a client-side JavaScript is used to perform this in ASP.NET.

But when the control you're using is one you've created yourself, you are responsible for programmatically generating the client-side JavaScript code that kicks off a form postback. To do this, the method that renders your control must include a reference to the GetPostBackEventReference method of the ASP.NET Page object. This method returns the name of a function (generated internally by ASP.NET) that is responsible for submitting the form to the page.

Listing 9.16 shows an example of a simple hyperlink server control that generates a client postback function through a call to GetPostBackEventReference.

Listing 9.16 Hyperlink Control That Can Generate Client-Side Postback of a Form

```
using System.Web.UI;
using System.Collections;
using System;

namespace CustomControls
{
  public class MyLinkButton : Control, IPostBackEventHandler
  {
    // Defines the Click event.
    public event EventHandler Click;

    // Invokes delegates registered with the Click event.

    protected virtual void  OnClick(EventArgs e)
    {
      if (Click != null)
      {
        Click(this, e);
      }

    }

    // Method of IPostBackEventHandler that raises change events.

    public void RaisePostBackEvent(String eventArgument)
    {
       OnClick(new EventArgs());
    }

    protected override void Render(HtmlTextWriter output)
    {
      output.Write(("<a  id ='" + this.UniqueID +
        "' href=\"javascript:" +
        Page.GetPostBackEventReference(this) + "\">"));
      output.Write((" " + this.UniqueID + "</a>"));
    }
  }
}


<%@ Register TagPrefix="Custom" Namespace="CustomControls"
Assembly = "CustomControls" %>

<script language="C#" runat="server">
  private void Button_Click(Object sender, EventArgs e)
  {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    TextBox.BackColor = System.Drawing.Color.LightGreen;
    TextBox.Text = "The link button caused postback.";
  }
</script>

<html>
<body>
  <form runat=server>
    Here is the custom link button.<br>
    <Custom:MyLinkButton Id = "Link"  OnClick = "Button_Click"
            runat=server/>
    <br><br>
    <asp:TextBox id = "TextBox" Text = "Click the link" Width = "200"
      BackColor = "Cyan" runat=server/>
    <br>
  </form>
</body>
</html>
```

Persistence Support

Your control has the capability to store state information that is posted back across round trips to the server. In English, this means that even though the page is destroyed and re-created each time the user submits a form to the server, your controls can maintain the values that users enter into them.

This is accomplished through the encoded postback data. You see this in ASP.NET Web forms programming all the time; if you set up a Web form with a bunch of text boxes and a button, and then fill in the text boxes and click the button to submit the form to the server, you should see that the contents of the text box remain the same, even though the page has been completely torn down and re-created as a result of its round trip to the server.

If the value of one or more of the properties of your control needs to stay the same, you can store the value of that property in the ViewState property of the control. Having a generic way to store state is useful because Web forms controls are created and destroyed each time a page is accessed.

The ViewState property is an object of type StateBag, found in the namespace System.Web.UI. It is a typical collection type, although it doesn't derive from any of the .NET framework collection types. It does, however, implement the interfaces IDictionary, ICollection, and IEnumerable, so you can use it as you would many other types of .NET collection objects.

To demonstrate this property, we'll create a simple counter control that has the capability to store, retrieve, and display a single numeric value. To provide the capability to store and retrieve the value consistently across page reloads, the control will store its property in the state bag collection provided by the Control base class.

Listing 9.17 provides the code for this server control.

Listing 9.17 Example of a Basic Server Control That Stores Property State

```
using System;
using System.Web;
using System.Web.UI;
```

This document is created with the unregistered version of CHM2PDF Pilot

```
namespace MyExamples
{
  public class Counter : Control
  {
    public Int32 CurrentValue
    {
      get
      {
        return (Int32)ViewState["CurrentValue"];
      }
      set
      {
        ViewState["CurrentValue"] = value;
      }
    }

    protected override void Render(HtmlTextWriter Output)
    {
      Output.Write("<table border='1' width='200'><tr>" +
        "<td align='center' bgcolor='#FFFF99'>" +
          this.CurrentValue + "</td></tr></table>");
    }

  }

}
```

Simple, simple, simple. You can see from the code example that the ViewState object is a simple key/value pair; the key can be whatever you want, although you'll probably want to give it the same name as the property it stores for simplicity's sake.

To test this control, compile it using the command line:

```
csc /t:library /out:counter.dll /r:System.dll /r:System.Web.dll counter.cs
```

After it's compiled and copied to the \bin directory, you can test it in a page similar to that shown in Listing 9.18. This page adds two ASP.NET Button controls to increment and decrement the value of your counter control.

Listing 9.18 Page to Contain the Counter Control

```
<%@ REGISTER TagPrefix="demo" Namespace="MyExamples" Assembly="counter" %>
<SCRIPT runat='server'>

  void UpButton_Click(Object Sender, EventArgs e)
  {
    Counter1.CurrentValue++;
  }

  void DownButton_Click(Object Sender, EventArgs e)
  {
   Counter1.CurrentValue--;
  }

</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>ASP.NET Page</TITLE>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  </HEAD>
  <BODY>
    <FORM runat='server'>
      <demo:counter id='Counter1' runat='server' />
      <asp:button id='DownButton' OnClick='DownButton_Click'
          text='Down' runat='server' /> 
      <asp:button id='UpButton' OnClick='UpButton_Click'
          text='Up' runat='server' />
    </FORM>
  </BODY>
</HTML>
```

When you load this page, you should be able to see that the Counter control's value is stored each time it's incremented. More importantly, it's displayed properly even if you hit the Refresh button on the browser—and the page designer didn't have to write any code to make that state retrieval happen.

You can see that this is the case if you rewrite the CurrentValue property, commenting out the lines of code that persist data to the state bag and instead storing the data in a private variable, as classes normally would be stored. Listing 9.19 shows an example of this.

Listing 9.19 Rewritten CurrentValue Property, Demonstrating Lack of State Persistence

```
private Int32 _CurrentValue;
public Int32 CurrentValue
{
    get
    {
      return _CurrentValue;
    }
    set
    {
    _CurrentValue = value;
    }
}
```

If you try this, don't forget to recompile counter.dll using the command line shown in the previous example. You should not have to make any changes to your page to use the new version of this control.

When your page uses this version of the control, the control does not have the capability to increment and decrement as it did previously. It can increment or decrement only once. This means the control can display only the values of 0 (the first time the page is loaded), 1 (when the Increment button is clicked), or -1 (when the Decrement button is clicked). This problem occurs because, in the absence of state persistence, the control's CurrentValue property is reinitialized to 0 every time the page is loaded. So, for example, when you click the Increment button, the form is first submitted to the server, destroying the page and the control along with it. The side effect is that the control's CurrentValue property is set to 0. Then the event procedure for the Increment button is run, setting the property to 1. But in the absence of persistent state, the control's property can never be incremented to 2 again, because when the page reloads, the control will be destroyed and reinitialized to 0 before the incrementing code has a chance to run.

As you take advantage of persistent property state, remember that a cost is associated with storing and retrieving state. Every time your control stores state information, that information is encoded and sent over the Internet from the client to the server and back again. The encoding process makes it easier for ASP.NET to handle your data, but it also has the side effect of making the postback data itself larger. Hence, postback will slow down your application

This document is created with the unregistered version of CHM2PDF Pilot

significantly if you overuse it.

Also remember that state information that is passed back and forth between client and server isn't encrypted, so it isn't secure. The information is encoded, but that isn't the same as encryption; it won't keep hackers' grimy mitts off your data.

NOTE

Ultimately, if your Web application passes sensitive information from client to server in any form, you should consider transferring the information using a Secure Socket Layer (SSL) connection. This way, it doesn't matter whether the postback information is encrypted, because under SSL, all the information that passes between client and server is encrypted. (There is a cost in terms of performance and configuration hassle associated with this, however, so plan carefully when you want to secure specific forms or pages.)

Building Validation Controls

You can build controls that have the capability to validate user input in Web forms. Such controls, called validators, are simply a type of ASP.NET server control.

NOTE

ASP.NET comes with a set of validation controls you can use to ensure that user input in Web forms controls is valid. Among these is a CustomValidator control that enables you to use any validation function you want. Before embarking on creating a new validation control from scratch, you may want to first determine whether the CustomValidator control will suit your needs.

Validation controls are discussed in Chapter 11.

Taking Advantage of Rich Clients

ASP.NET server controls have the capability to tailor their output to the capabilities of the client. In the case of present-day Web applications, this capability enables you to write applications that take advantage of features, such as Dynamic HTML, that are supported by only the most advanced browsers, without your having to either write your pages twice or restrict your user base to a particular browser (usually Internet Explorer). This is called uplevel/downlevel rendering.

Uplevel/downlevel rendering is a nice feature that can result in performance increases in situations such as client-side data validation (discussed in Chapter 11), and the server controls you design yourself can also take advantage of

This document is created with the unregistered version of CHM2PDF Pilot

uplevel/downlevel rendering functionality in ASP.NET the same way that ASP.NET's own server controls do.

NOTE

When Microsoft documentation refers to a "rich client," it's really talking about Internet Explorer 5.0 running on Windows. When it talks about an "uplevel browser," it's talking about the same thing. Conversely, when it's talking about a "downlevel browser," it's talking about every browser except Internet Explorer (including, but not limited to, Netscape Navigator).

I don't like using Netscape if I don't have to, particularly to test Web sites I'm developing, because it doesn't seem to run as fast as Internet Explorer. If you want to test uplevel/downlevel functionality but don't want to use Netscape, you have an alternative. The Opera browser is fast, lightweight, aggressively W3C standards–compliant, and free, if you don't mind looking at a few advertisements while you browse (you can get an ad-free version of Opera by paying a registration fee). It's great for testing Web applications when you need an alternative to Internet Explorer. Download Opera from http://www.opera.com.

One way ASP.NET provides uplevel/downlevel rendering is through HtmlTextWriter object, which is passed as an argument to the Render method of the Control class. This object normally renders HTML 4.0, but is swapped out by the Page object in favor of the System.Web.UI.Html32TextWriter class, which automatically renders HTML 3.2 where appropriate.

Supporting Designers in Custom Server Controls

If you want developers to be able to work with your server controls in visual development environments such as Visual Studio .NET, you should add support for visual designers to your control. A designer is a package of information that determines how your control interacts with the development environment when a developer is working with it.

An important distinction exists between the way a control behaves when it's being used by a developer in a tool such as Visual Studio .NET—known as design time—and the way a control executes in a running application, known as runtime. This distinction will be familiar to you if you've created ActiveX controls in previous versions of Visual Basic or Visual C++.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 10. Using XML

IN THIS CHAPTER

- What Is XML?
- Accessing XML Data Using .NET Framework Classes
- Defining and Validating XML with Schemas
- Processing XML Documents Using Style Sheets
- Class Reference

In the .NET framework, XML is very important. It serves as the foundation for many of the .NET technologies. Database access is XML based in ADO.NET. Remote interoperability, known as XML Web services or SOAP, is also XML based. It is true that many of the implementation details of XML are hidden inside objects or inside the Visual Studio .NET development environment. But for tasks such as debugging, interoperability with other platforms, performance analysis, and your own peace of mind, it still makes sense for you as a .NET developer to have a handle on what XML is, how it works, and how it is implemented in the .NET framework. This chapter will help with that objective.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

What Is XML?

Here's a problem you've probably faced before. A customer or colleague comes to you asking for help working with an application that was written five years ago. Nobody who originally worked on the application still works for the company; the original developer died in a bizarre gardening accident some years back. The customer wants you to write a Web-based reporting system to handle the data emitted by this dinosaur application.

You now have the unenviable task of figuring out how this thing works, parsing the data it emits, and arranging that data in some recognizable format—a report.

Let's assume that the developer of the original application attempted to make it easy on you by expressing the data in some standardized format—maybe one in which elements within rows of data are separated from each other by a designated character, such as a comma or a tab. This is known as a delimited format. Listing 10.1 demonstrates a comma-delimited document.

Listing 10.1 A Comma-Delimited Document

```
Jones,Machine Gun,401.32,New York
Janson,Hand Grenade,79.95,Tuscaloosa
Newton,Artillery Cannon,72.43,Paducah
```

However, a few problems occur with the delimited format. First, what happens if the data itself contains a comma or a tab? In this case, you're forced to use a more complicated delimiter—typically a comma with data enclosed in quotation marks. That different documents can use different delimiters is a problem in itself, though. There's no such thing as a single universal parse algorithm for delimited documents.

To make it even more difficult, different operating systems have different ideas about what constitutes the end of a line. Some systems (such as Windows) terminate a line with a carriage return and a line feed (ASCII 13 and 10, respectively), whereas others (such as Unix) just use a line feed.

Another problem: What is this data? Some of it, such as the customer's name and the item, is obvious. But what does the number 401.32 represent? Ideally, we want a document that is self-describing—one that tells us at a glance what all the data represents (or at least gives us a hint).

Another big problem with delimited documents: How can you represent related data? For example, it might be nice to view all the information about customers and orders in the same document. You can do this with a delimited document, but it can be awkward. And if you've written a parser that expects the first field to be the customer name and the fourth field to be the product name, adding any new fields between them breaks the parser.

Internet technology mavens realized that this scenario is frighteningly common in the world of software development—particularly in Internet development. XML was designed to replace delimited data, as well as other data formats, with something standard, easy to use and to understand, and powerful.

This document is created with the unregistered version of CHM2PDF Pilot

Advantages of XML

In a networked application, interoperability between various operating systems is crucial; the transfer of data from point A to point B in a standard, understandable way is what it's all about. For tasks that involve parsing data, then, using XML means spending less time worrying about the details of the parser itself and more time working on the application.

Here are some specific advantages of XML over other data formats:

- XML documents are easily readable and self-describing—Like HTML, an XML document contains tags that indicate what each type of data is. With good document design, it should be reasonably simple for a person to look at an XML document and say, "This contains customers, orders, and prices."
- XML is interoperable—Nothing about XML ties it to any particular operating system or underlying technology. You don't have to ask anyone's permission or pay anyone money to use XML. If the computer you're working on has a text editor, you can use it to create an XML document. Several types of XML parsers exist for virtually every operating system in use today (even really weird ones).
- XML documents are hierarchical—It's easy to add related data to a node in an XML document without making the document unwieldy.
- You don't have to write the parser—Several types of object-based parser components are available for XML. XML parsers work the same way on virtually every platform. The .NET platform contains support for the Internet-standard XML Document Object Model, but Microsoft has also thrown in a few XML parsing widgets that are easier to use and that perform better than the XML DOM; we'll cover these later in this chapter.
- Changes to your document won't break the parser—Assuming that the XML you write is syntactically correct, you can add elements to your data structures without breaking backward compatibility with earlier versions of your application.

Is XML the panacea to every problem faced by software developers? XML won't wash your car or take out the garbage for you, but for many tasks that involve data, it's a good choice.

At the same time, Visual Studio .NET hides much of the implementation details from you. Relational data expressed in XML is abstracted in the form of a DataSet object. XML schemas (a document that defines data types and relationships in XML) can be created visually, without writing code. In fact, Visual Studio .NET can generate XML

This document is created with the unregistered version of CHM2PDF Pilot

schemas for you automatically by inspecting an existing database structure.

XML Document Structure and Syntax

XML documents must adhere to a standard syntax so that automated parsers can read them. Fortunately, the syntax is pretty simple to understand, especially if you've developed Web pages in HTML. The XML syntax is a bit more rigorous than that of HTML, but as you'll see, that's a good thing. There are a million ways to put together a bogus, sloppy HTML document, but the structure required by XML means that you get a higher level of consistency; no matter what your document contains, the rules that govern how an XML document can be parsed are the same.

Declaration

The XML declaration is the same for all XML documents. An XML declaration is shown in Listing 10.2.

Listing 10.2 XML 1.0 Declaration

```
<?xml version="1.0"?>
```

The declaration says two things: This is an XML document (duh), and this document conforms to the XML 1.0 W3C recommendation (which you can get straight from the horse's mouth at http://www.w3.org/TR/REC-xml). The current and only W3C recommendation for XML is version 1.0, so you shouldn't see an XML declaration that's different from what's in Listing 10.2. But you might in the future, when the specification is revised into new versions.

NOTE

A W3C recommendation isn't quite the same as a bona fide Internet standard, but it's close enough for our purposes.

The XML declaration, when it exists, must exist on the first line of the document. The declaration does not have to exist, however; it is an optional part of an XML document. The idea behind a declaration is that you may have some automated tool that trawls document folders looking for XML. If your XML files contain declarations, it'll be much easier for such an automated process to locate XML documents (as well as to differentiate them from other marked-up documents, such as HTML Web pages).

Don't sweat it too much if you don't include a declaration line in the XML documents you create. Leaving it out doesn't affect how data in the document is parsed.

Elements

An element is a part of an XML document that contains data. If you're accustomed to database programming or working with delimited documents, you can think of an element as a column or a field. XML elements are sometimes also called nodes.

This document is created with the unregistered version of CHM2PDF Pilot

XML documents must have at least one top-level element to be parsable. Listing 10.3 shows an XML document with a declaration and a single top-level element (but no actual data).

Listing 10.3 A Simple XML Document with a Declaration and a Top-Level Element

```
<?xml version="1.0"?>
<ORDERS>
</ORDERS>
```

This document can be parsed, even though it contains no data. Note one important thing about the markup of this document: It contains both an open tag and a close tag for the <ORDERS> element. The closing tag is differentiated by the slash (/) character in front of the element name. Every XML element must have a closing tag—lack of a closing tag will cause the document to be unparsable. The XML declaration is the only part of an XML document that does not require a closing tag.

This is an important difference between XML and HTML. In HTML, some elements require close tags, but many don't. Even for those elements that don't contain proper closing tags, the browser often attempts to correctly render the page (sometimes with quirky results).

XML, on the other hand, is the shrewish librarian of the data universe. It's not nearly as forgiving as HTML and will rap you on the knuckles if you cross it. If your XML document contains an element that's missing a close tag, the document won't parse. This is a common source of frustration among developers who use XML. Another kicker is that, unlike HTML, tag names in XML are case sensitive. This means that <ORDERS> and <orders> are considered to be two different and distinct tags.

Elements That Contain Data

The whole purpose of an XML element is to contain pieces of data. In the previous example, we left out the data. Listing 10.4 shows an evolved version of this document, this time with data in it.

Listing 10.4 An XML Document with Elements That Contain Data

```
<?xml version="1.0"?>
<ORDERS>
  <ORDER>
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <ID>33849</ID>
    <CUSTOMER>Steve Farben</CUSTOMER>
    <TOTALAMOUNT>3456.92</TOTALAMOUNT>
  </ORDER>
</ORDERS>
```

If you were to describe this document in English, you'd say that it contains a top-level ORDERS element and a single ORDER element, or node. The ORDER node is a child of the ORDERS element. The ORDER element itself contains four child nodes of its own:

DATETIME, ID, CUSTOMER, and TOTALAMOUNT.

This document is created with the unregistered version of CHM2PDF Pilot

Adding a few additional orders to this document might give you something like Listing 10.5.

Listing 10.5 An XML Document with Multiple Child Elements Beneath the Top-Level Element

```
<?xml version="1.0"?>
<ORDERS>
  <ORDER>
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <ID>33849</ID>
    <CUSTOMER>Steve Farben</CUSTOMER>
    <TOTALAMOUNT>3456.92</TOTALAMOUNT>
  </ORDER>
  <ORDER>
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <ID>33856</ID>
    <CUSTOMER>Jane Colson</CUSTOMER>
    <TOTALAMOUNT>401.19</TOTALAMOUNT>
  </ORDER>
  <ORDER>
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <ID>33872</ID>
    <CUSTOMER>United Disc, Incorporated</CUSTOMER>
    <TOTALAMOUNT>74.28</TOTALAMOUNT>
  </ORDER>
</ORDERS>
```

Here's where developers sometimes get nervous about XML. With a document like this, you can see that there's far more markup than data. Does this mean that all those extra bytes will squish your application's performance?

Maybe, but not necessarily. Consider an Internet application that uses XML on the server side. When this application needs to send data to the client, it first opens and parses the XML document (we'll discuss how XML parsing works later in this chapter). Then some sort of result—in all likelihood, a tiny subset of the data, stripped of markup—will be sent to the client Web browser. The fact that there's a bunch of markup there doesn't slow the data transfer down significantly.

At the same time, there is a way to express data more succinctly in an XML document, without the need for as many open and closing markup tags. You can do this through the use of attributes.

Attributes

An attribute is another way to enclose a piece of data in an XML document. An attribute is always part of an element; it typically modifies or is related to the information in the node. In a relational database application that emits XML, it's common to see foreign key data expressed in the form of attributes.

For example, a document that contains information about a sales transaction might use attributes as shown in Listing 10.6.

Listing 10.6 An XML Document with Elements and Attributes

```
<?xml version="1.0"?>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
<ORDERS>
  <ORDER id="33849" custid="406">
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <TOTALAMOUNT>3456.92</TOTALAMOUNT>
  </ORDER>
</ORDERS>
```

As you can see from the example, attribute values are always enclosed in quotation marks. Using attributes tends to reduce the total size of the document (because you don't need to store open and close tags for the element). This has the effect of reducing the amount of markup at the expense (in some cases) of readability. Note that you are allowed to use either single or double quotation marks anywhere XML requires quotes.

This element/attribute syntax may look familiar from HTML, which uses attributes to assign values to elements the same way XML does. But remember that XML is a bit more rigid than HTML; a bracket out of place or a mismatched close tag will cause the entire document to be unparsable.

Enclosing Character Data

At the beginning of this chapter, we discussed the various dilemmas involved with delimited files. One of the problems with delimiters is that if the delimiter character exists within the data, it's difficult or impossible for a parser to know how to parse the data.

This problem is not confined to delimited files; XML has similar problems with containing delimiter characters. The problem arises because the de facto XML delimiter character (in actuality, the markup character) is the left angle bracket, also known as the less-than symbol. In XML, the ampersand character (&) can also throw off the parser.

You've got two ways to deal with this problem in XML: Either replace the forbidden characters with character entities or use a CDATA section as a way to delimit the entire data field.

Using Character Entities

You might be familiar with character entities from working with HTML. The idea is to take a character that might be interpreted as a part of markup and replace it with an escape sequence to prevent the parser from going haywire. Listing 10.7 provides an example of this.

Listing 10.7 An XML Document with Escape Sequences

```
<?xml version="1.0"?>
<ORDERS>
  <ORDER id="33849">
    <NAME>Jones &amp; Williams Certified Public Accountants</NAME>
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <TOTALAMOUNT>3456.92</TOTALAMOUNT>
  </ORDER>
</ORDERS>
```

Take a look at the data in the NAME element in the code example. Instead of an ampersand, the & character entity is used. (If a data element contains a left bracket, it should be escaped with the < character entity.)

This document is created with the unregistered version of CHM2PDF Pilot

When you use an XML parser to extract data with escape characters, the parser will automatically convert the escaped characters to their correct representation.

Using CDATA Elements

An alternative to replacing delimiter characters is to use CDATA elements. A CDATA element tells the XML parser not to interpret or parse characters that appear in the section.

Listing 10.8 demonstrates an example of the same XML document from before, this time delimited with a CDATA section rather than a character entity.

Listing 10.8 An XML Document with a CDATA Section

```
<?xml version="1.0"?>
<ORDERS>
  <ORDER id="33849">
    <NAME><![CDATA[Jones & Williams Certified Public Accountants]]></NAME>
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <TOTALAMOUNT>3456.92</TOTALAMOUNT>
  </ORDER>
</ORDERS>
```

In this example, the original data in the NAME element does not need to be changed, as in the previous example. Here, the data is wrapped with a CDATA element. The document is parsable, even though it contains an unparsable character (the ampersand).

Which technique should you use? It's really up to you. You might prefer to use the CDATA method because it doesn't require altering the original data, but it has the disadvantage of adding a dozen or so bytes to each element.

Abbreviated Close-Tag Syntax

For elements that contain no data, you can use an abbreviated syntax for element tags to reduce the amount of markup overhead contained in your document. Listing 10.9 demonstrates this.

Listing 10.9 An XML Document with Empty Elements

```
<?xml version="1.0"?>
<ORDERS>
  <ORDER id="33849" custid="406">
    <DATETIME>1/4/2000 9:32 AM</DATETIME>
    <TOTALAMOUNT />
  </ORDER>
</ORDERS>
```

You can see from the example that the TOTALAMOUNT element contains no data. As a result, we can express it as <TOTALAMOUNT /> instead of <TOTALAMOUNT> </TOTALAMOUNT>. It's perfectly legal to use either syntax in your XML documents; the abbreviated syntax is generally better, though, because it reduces the size of

This document is created with the unregistered version of CHM2PDF Pilot

your XML document.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Accessing XML Data Using .NET Framework Classes

Now that you've seen how to create an XML document, we get to the fun part: how to write code to extract and manipulate data from an XML document using classes found in the .NET frameworks. There's no one right way to do this; in fact, before .NET came along, two predominant ways were used to parse an XML document: the XML Document Object Model (DOM) and Simple API for XML (SAX).

An implementation of the XML DOM exists in the .NET framework. However, in this chapter we'll primarily focus on .NET's own XML handlers, such as the XmlNodeReader, XmlTextReader, and XmlTextWriter objects. These objects are the standard .NET way to access XML data; they provide a good combination of high performance, .NET integration, and ease of programming. But you should know about the other ways to deal with XML, too—particularly because the specialized .NET reader and writer objects are designed to interact with the Internet-standard DOM objects. So for the remainder of this chapter, we'll include brief examples of how to work with the DOM model, as well.

About Simple API for XML (SAX)

Simple API for XML (SAX) was designed to provide a higher level of performance and a simpler programmability model than XML DOM. It uses a fundamentally different programmability model. Instead of reading in the entire document at once and exposing the elements of the document as nodes, SAX provides an event-driven model for parsing XML.

SAX is not supported in .NET—yet. In fact, it's not even an official Internet standard. It's a programming interface for XML that was created by developers who wanted an XML parser with higher performance and a smaller memory footprint, especially when parsing very large documents.

If you are currently writing applications using SAX and want to use SAX in your .NET applications today, you can do so by using the MSXML 3.0 COM library through the COM interoperability features in .NET.

NOTE

Although it is not yet supported in the .NET framework, SAX is supported in Microsoft's COM-based XML parser implementation. For more information on this tool, see http://msdn.microsoft.com/xml/.

Using the XML Document Object Model

The XML Document Object Model (DOM) is a programming interface used to parse XML documents. It was the first programming interface provided for XML by Microsoft; XML DOM implementations are available that target other languages and other operating systems.

This document is created with the unregistered version of CHM2PDF Pilot

The original Microsoft XML DOM implementation is COM based, so it is accessible from any COM-compliant language. The XML parsers in .NET are, naturally, accessible from any .NET-compliant language.

The XML DOM does its magic by taking an XML document and exposing it in the form of a complex object hierarchy. This kind of hierarchy may be familiar to you if you've done client-side HTML Document Object Model programming in JavaScript or VBScript. The number of objects in XML DOM is fairly daunting; no fewer than 20 objects are in the base implementation, and then the Microsoft implementation adds a number of additional interfaces and proprietary extensions.

Fortunately, the number of objects you need to work with on a regular basis in the XML DOM is minimal. In fact, the XML DOM recommendation segregates the objects in the DOM into two groups: fundamental classes and extended classes. Fundamental classes are the ones that application developers find most useful; the extended classes are primarily useful to tools developers and people who like to pummel themselves with detail.

The fundamental classes of the XML DOM as implemented in the .NET framework are XmlNode, XmlNodeList, and XmlNamedNodeMap. These classes, as well as the parent XmlDocument class, are illustrated in Figure 10.1.

Figure 10.1. Fundamental XML DOM objects.

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Defining and Validating XML with Schemas

Whenever you use or manipulate data, you need to have a way of answering certain questions about that data. Is an Invoice ID stored as a text value or a numeric value? Is a phone number limited to 10 digits? For that matter, can a person have more than one phone number? What happens if the person has none?

All these questions have to do with the concepts of data definition and validation. Application developers have historically embedded validation logic in application code. Sophisticated designs can encapsulate validation logic in various ways, but in most cases, the data definition and validation logic aren't accessible to processes outside of your application. This defeats the purpose of XML on a number of levels. Remember that XML is designed to be interoperable and human readable. When you commit validation logic to code, you've almost inherently made the validation logic inaccessible to other processes that might come along later. It is, in essence, a black box. The concept of encapsulating data validation in a class is a good and useful thing, but if other developers can't easily access your data design from outside sources, it may not be as useful to them.

A way exists to express and validate data designs expressed in XML. This is done through a standard descriptive format referred to as XML schemas (sometimes abbreviated XSD). Because the various .NET tools provide good support for XML schemas, we'll devote some time to discussing how schemas work, how to build them, and what you can do with them in your Internet applications.

About Document Type Definitions (DTDs)

The first technology used for validating XML structures was known as Document Type Definition (DTD). By linking a DTD document to an XML file, you can ensure that the XML document contains valid data types and structure.

The problem with DTDs is that they have limitations with respect to the things they can do. One glaring limitation is that DTDs can't define data types of elements that appear in a document.

But the most damning implication of using DTDs to validate XML is that DTDs are written using a syntax that is completely removed from that of XML itself; if you want to validate your data using the DTD format, you must ascend a learning curve.

A good example of a DTD is the DTD for XML itself, which resides at http://www.w3.org/XML/1998/06/xmlspec-v21.dtd. By looking at this DTD, you can get a sense for how different the syntax of DTD is. Indeed, the response of many developers who had to use DTDs in the early days of XML was, "Couldn't we use XML syntax to define the structure of an XML document?"

Microsoft chose to use a more evolved document definition technology for XML in the .NET universe—the XML schema. The most important benefit of XML schemas is that you can write an XML schema using the XML syntax you presumably already know. For these reasons, this chapter focuses on schemas rather than DTDs.

This document is created with the unregistered version of CHM2PDF Pilot

NOTE

The Visual Studio .NET development environment gives developers a graphical way to build XML schemas and contains little or no support for DTDs. You can certainly use DTDs with XML data in the .NET framework; you just won't get much help from the tools.

Before we proceed, it's worth noting that you're never required to validate the XML documents that you use in your application development; XML documents can live out their whole lives without ever knowing or using an XML schema. However, it's a good idea to validate them for the sake of consistency. In addition, certain tools (including Visual Studio .NET) use XML schemas in various interesting and useful ways. Having a handle on what an XML schema is and how it works will give you a leg up on using these tools.

NOTE

The official W3C documentation on XML schemas comes in three parts: a primer, which as of this writing runs 73 printed pages, and two sections that document the schema specification in detail. The whole thing starts at http://www.w3.org/XML/Schema.

About XML Data-Reduced Schemas

The COM-based Microsoft XML implementation that existed before the arrival of the .NET framework used a syntax known as XML Data-Reduced (XDR) schemas. Confusingly, the Microsoft documentation refers to XDR as "XML Schemas," even though that's really a different, albeit related, syntax from that provided by the W3C. The .NET tools support both the XDR and W3C way of expressing schemas, but Visual Studio .NET follows the W3C schema syntax, so you'll likely see the W3C syntax used in .NET applications more often.

We are including this section on XDR schemas so that you can identify and understand the difference between XDR and W3C XML schemas, and as a way to make it easier for users of the MSXML library who are moving to the .NET framework to migrate their applications. Also, because the XML-handling objects in the .NET framework can process XDR-validated documents, it's possible that you will need to use XDR at some point, even though it's being superseded by the W3C schema format.

Listing 10.32 shows an example of a complete XDR schema definition.

Listing 10.32 Example of an XDR Schema

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">

  <ElementType name='TITLE' content='textOnly' />

  <AttributeType name='IDType' dt:type='integer' />
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  <ElementType name='AUTHOR' content='textOnly'>
    <attribute type='IDType' />
  </ElementType>

  <ElementType name='BOOK' content='mixed'>
    <element type = 'TITLE' />
    <element type = 'AUTHOR' />
  </ElementType>

</Schema>
```

The example begins with a Schema node, to indicate that this is the start of a schema. The two xmlns attributes refer to external schema documents; the first one is for XML itself, the second one is for the data types defined by the Microsoft data types defined for use in XDR schemas.

The ElementType nodes in the schema document form the definition of the nodes that compose the document. In this example, you can see two types of ElementTypes defined; a simple type (containing no child nodes, such as TITLE) and a complex type (containing child nodes and/or attributes, such as BOOK). We'll discuss simple and complex types in more detail in the section "Understanding Simple and Complex Types" later in this chapter.

NOTE

The Microsoft reference on XDR schemas is at http://msdn.microsoft.com/xml/reference/schema/start.asp. Another useful Microsoft link is the XML Schema Developer's Guide, located at http://msdn.microsoft.com/xml/xmlguide/schema-overview.asp.

This reference material was created to document the behavior of the MSXML parser found in Internet Explorer 5.0. It may not have direct applicability to XML applications that you build using the .NET tools (use the W3C specification for XML schemas you build in .NET). Note, again, that when Microsoft refers to "XML Schema," it may be referring to either XDR schemas or W3C-style XML schemas. In general, what you get in the .NET tools are true W3C XML schemas.

This section is intended to give you the briefest example of an XDR schema so you can understand what an XDR schema looks like. Because the Visual Studio .NET uses the more recent W3C recommendation for XML schemas, however, we'll spend the rest of this section discussing the W3C syntax for XML document definition and validation.

NOTE

Using a tool that comes with the .NET framework SDK, you can convert existing XDR schemas to the W3C format described in the next section. Known as the XML Schema Definition Tool (xsd.exe), this command-line tool can also create basic schemas from existing XML files and build ADO.NET classes in Visual Basic.NET or C# from existing schemas.

This document is created with the unregistered version of CHM2PDF Pilot

Creating W3C XML Schemas

A W3C XML schema is conceptually similar to an XDR schema, but has a number of implementation differences. Because XML schema is on its way to becoming an Internet standard, it's better to use the W3C standard format because you can expect a better level of interoperability as the standard propagates. Fortunately, the new XML-handling tools included in the .NET framework and Visual Studio .NET tend to use the newer W3C versions of technologies such as schemas, so you'll have lots of help when building applications that are designed to be Internet standard and interoperable.

Our objective in this section is to perform a very simple validation on a simple XML document. Because the W3C XML schema language is a very complex syntax that could warrant a short book of its own, in this section we'll cover only the basics of XML schema—particularly, how to create a schema that validates an XML document using the XML-handling objects in the .NET framework classes.

Listing 10.33 shows the document we'll be validating in this section.

Listing 10.33 Simplified book.xml Document

```
<BOOK isbn="1234567890">
  <TITLE>Little Red Riding Hood</TITLE>
  <AUTHOR>Dave-Bob Grimm</AUTHOR>
</BOOK>
```

As you can see, this is a greatly simplified version of the books.xml document we've used in examples throughout this chapter. Rather than containing an unlimited number of books, this document contains only a single book, so it might be used in situations where one software process hands off book information to another.

Like XDR schemas, W3C schemas are generally linked to external files that provide the basic definition of what a schema is. As a result, the W3C-compliant schemas you create will typically begin with a reference to the standard W3C schema definition (known as the schema of schemas). The W3C schema definition gives your schema access to basic data types and structures you'll need to construct schemas to define and validate your XML documents. This basic definition is shown in Listing 10.34.

Listing 10.34 W3C Schema Definition Boilerplate

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- Your schema definition goes here -->

</xsd:schema>
```

This is a boilerplate definition that you'll probably include in most, if not all, of your XML schema documents. (Note that the XML schema-editing function provided by Visual Studio .NET generates a slightly different boilerplate schema definition; what you see here is more streamlined.)

This document is created with the unregistered version of CHM2PDF Pilot

This boilerplate provides the same basic function as the initial definition of the XDR schema shown in a previous example, but it provides a different basic schema type and associates it with the xsd namespace. This means that you'll often see elements of a W3C schema prefixed with the xsd namespace; this is done to prevent namespace collisions between elements defined by the xsd schema definition and elements in your documents with the same name.

The next step to defining your own W3C schema is to define the data types that can appear in your document. To do this, you must have a handle on simple and complex data types in XML and how the W3C XML schema defines them.

Understanding Simple and Complex Types

As you know, an XML document is inherently hierarchical. Every XML document is composed of nodes that can contain child nodes, nested as deeply as necessary to represent a given data structure. When you're authoring an XML schema, you need to be able to make a distinction between simple and complex types. A complex type is defined as any node that has children or attributes; a simple type has no children or attributes. For example, in the book.xml document used as an example in Listing 10.33, BOOK is a complex type because it contains two child elements, AUTHOR and TITLE, as well as an attribute, isbn. AUTHOR, on the other hand, is a simple type, because it contains nothing but a text string (the name of the book's author).

The distinction between simple and complex types becomes important when building XML schemas because the two types are described in different ways in the schema format. In XML schema authoring, it's common to define the simple types first and the complex types later because the complex types are almost invariably built on the simple type definitions.

Listing 10.35 shows an example of a schema with a simple type definition.

Listing 10.35 W3C Schema Containing a Definition for a Simple Type

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="ISBNType">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="10"/>
    </xsd:restriction>
  </xsd:simpletype>

</xsd:schema>
```

The first few lines of this schema definition are the same as the previous example; they refer to the external master schema maintained by the W3C that defines basic data types and so forth.

The xsd:simpleType node contains the definition for a simple type called ISBNType. You can see from the definition that this type contains a restriction, which provides additional information about the data type and (optionally) the nature of the data that this data type supports. In this case, ISBNType is declared to be a string that can have a maximum length of 10 characters. (Note that although ISBN stands for International Standard Book Number, an ISBN can contain alphabetic characters in addition to numbers, so we declare it to be a string type.)

This document is created with the unregistered version of CHM2PDF Pilot

Be careful typing the names of attributes such as maxLength in your XML schema definitions. As with all XML elements, the elements of an XML schema are case sensitive.

You don't have to define named types in your XML schemas. The advantage is reusability—after you've defined a named type, you can reuse it anywhere you like in the schema definition (or refer to the schema from another schema and reuse it that way). When used in this way, XML schemas behave a bit like class definitions.

After you've created a type definition, you can declare that your document will contain elements of this type. Listing 10.36 shows an example of a schema containing a reference to the ISBNType simple type.

Listing 10.36 W3C Schema Containing an Element Definition That Refers to a Type Definition

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="ISBN" type="ISBNType"></xsd:element>
    <xsd:simpleType name="ISBNType">
        <xsd:restriction base="xsd:string">
            <xsd:maxLength value="10" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

The element definition indicates that XML documents based on this schema will contain an element named ISBN. The type definition for this element is the ISBNType type created in the previous section.

This schema would be sufficient if we were interested only in creating XML documents containing lists of ISBN numbers. But the book information we're working with contains much more than that—we need to transmit the title and author of the book as well. To do this, we'll need to modify our schema to include a new complex type, called BookType, that defines TITLE and AUTHOR elements, as well as an isbn attribute. The isbn attribute is defined as an ISBNType, which means it takes on the properties of that type definition; it's a string data type with a maximum length of 10 characters.

Listing 10.37 shows another version of the schema, this time with a more complete definition of the BookType type. This time, we've added TITLE and AUTHOR types to the BookType.

Listing 10.37 W3C Schema Containing a Complex Type That Refers to a Simple Type

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Element definition -->
    <xsd:element name="BOOK" type="BookType"></xsd:element>

    <!-- Complex type definition -->
    <xsd:complexType name="BookType">
        <xsd:all>
            <xsd:element name="TITLE" type="xsd:string" />
            <xsd:element name="AUTHOR" type="xsd:string" />
        </xsd:all>
        <xsd:attribute name="isbn" type="ISBNType" />
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    </xsd:complexType>

    <!-- Simple type definition with restriction-->
    <xsd:simpleType name="ISBNType">
        <xsd:restriction base="xsd:string">
            <xsd:maxLength value="10" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

This version of the schema completes the complex type definition BookType by adding two elements: TITLE and AUTHOR. Both elements are defined as conventional strings with no special validation logic attached. The <xsd:all> element indicates that two or more child elements can appear in any order beneath the parent element. If you need to specify that the child elements should appear in a particular order, use <xsd:sequence> instead of <xsd:all>.

The big change in this final version of the schema is the addition of the BOOK element definition. Because the type of book was defined previously as a complex type, this section is very straightforward; all we need to do is reference the BookType complex type.

You'll notice, too, that this version of the schema contains comments; comments in XML are syntactically identical to comments in HTML. You should include comments in any XML file you create wherever there's a chance that somebody who comes along later might misunderstand what's going on in your document.

Validating Documents Using W3C Schemas

To validate an XML document using a schema, you first create the schema and then link the schema to an XML document defined by the schema. When an XML parser processes a document that is linked to a schema, the parser will first download the schema document(s) associated with that file. If the file fails to conform with any of the rules specified in the schema, the parser will complain (and in most cases refuse to proceed).

Schemas are often contained in a file separate from the XML data file. This enables you to change the schema definition without having to slog through the data itself. By placing a schema file in a location accessible through the Internet, any file anywhere can access and utilize the data structure and validation rules found in the schema.

In addition to linking XML documents to schemas, schemas themselves can be linked to other schemas. This gives you the capability to build progressively more sophisticated schemas based on more basic schema definitions created in the past.

After you've created an XML schema and linked it to an associated XML data document that implements the schema, you should test it to ensure that it does what you want. Because Internet Explorer understands XML and can render XML in the browser, you can use Internet Explorer 5.0 or later to determine whether your XML document parses. To do this, simply load the file into Internet Explorer using the URL text box or by using the File, Open menu command.

As with any XML parser, Internet Explorer automatically downloads the schema definition when it encounters an

This document is created with the unregistered version of CHM2PDF Pilot

XML document that is linked to a schema. After the file and the external schemas are downloaded, the browser then attempts to parse the document. If parsing is successful, Internet Explorer displays the document. If it's unsuccessful, it usually gives you an error message in the browser.

Problems with XML rendering and parsing in a validated context usually stem from one of two problems: The document is not well formed (meaning the document is lacking an end tag for a node, for example), or the document is invalid (according to the validation rules defined in the schema file).

XML parsers almost invariably reject documents that are not well formed; whether they reject invalid documents depends on the tool you use to handle the document. For example, an XmlTextReader object will not throw an error when it reads an invalid XML document, but the XmlValidatingReader object will. The XmlValidatingReader object is introduced in the next section, "Using .NET Framework Objects to Validate XML Schemas."

NOTE

You should watch out for a couple of things when working with XML document validation with schemas. First, make sure your computer has a connection to the Internet when you load a validated document, because schemas must typically access other dependent schemas, and the way that's most commonly done is by downloading them over the Net.

Next, remember that XML is case sensitive—uppercase and lowercase matter. Spelling an element name BOOK in one place in the document and then attempting to refer to something called Book or book later on will cause problems.

Using .NET Framework Objects to Validate XML Schemas

Earlier in this chapter, we discussed how to read a document using the XmlTextReader object provided by the .NET framework. You can use an XML schema to validate a document when reading a document using the .NET framework. To do this, you use the XmlValidatingReader object.

NOTE

The XmlValidatingReader class is found in the System.Xml namespace. Like the XmlTextReader object described earlier in this chapter, XmlValidatingReader inherits from System.Xml.XmlReader. A reference to the classes, properties, and methods introduced in this chapter is included at the end of this chapter.

Because they both inherit from the same base class, the XmlValidatingReader object works similar to the XmlTextReader object. To validate a document using the XmlValidatingReader object, set the object's Validation property to one of the values enumerated in System.Xml.ValidationType. The Validation property can be set to one of these values:

This document is created with the unregistered version of CHM2PDF Pilot

- ValidationType.None (no validation)
- ValidationType.DTD (use a document type definition for validation)
- ValidationType.Schema (use an XSD schema)
- ValidationType.XDR (use an XDR schema)
- ValidationType.Auto (infer one of the preceding values)

The value ValidationType.Auto tells the XmlValidatingReader object to infer which type of schema to use based on what the document contains. This means that it's possible for no validation to occur when using the Auto type—if the XML document does not actually contain a link to a schema or DTD and the validation type is set to Auto, no validation will occur. For this reason, it's a good idea to explicitly set the validation type (if you know what it's going to be).

If the XML document does not contain a link to a schema, you can add a link to a schema programmatically. Do this by using the Add method of the Schemas collection contained by the XmlValidatingTextReader object.

After you set the validation type, assign a schema, you must then write code to actually perform the validation. This is similar to the code you write. Listing 10.38 shows an example of this.

Listing 10.38 XML Validation Subroutine Using the XmlValidatingReader Object

```
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Schema" %>
<SCRIPT runat='server'>

    void Page_Load(Object Sender,EventArgs e)
   {
      XmlTextReader tr = new XmlTextReader(Server.MapPath("book-invalid.xml"));
      XmlValidatingReader vr  = new XmlValidatingReader(tr);

      vr.ValidationType = ValidationType.Schema;
      vr.Schemas.Add(null, Server.MapPath("book.xsd"));

      while(vr.Read())
      {
     Response.Write("[" + vr.Name + "]" + vr.Value + "<BR>");
     if(vr.NodeType == XmlNodeType.Element)
     {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        while(vr.MoveToNextAttribute())
      {
        Response.Write("[" + vr.Name + "]" + vr.Value + "<BR>");
      }
     }
       }
    }

</SCRIPT>
```

This code throws an error when it encounters an element of the document that violates the schema. The code will fail because it parses a version of book.xml that contains a validation error (an ISBN that is too long).

Raising errors when XML schema validation rules are broken is fine, but you may want a more granular level of control over how the document is validated, in addition to richer information on where validation errors were encountered. To do this, you can cause the XmlValidatingReader object to raise events when it encounters validation problems in the documents it parses.

To handle the events raised by an XmlValidatingReader object, you must create an event-handling procedure in your code and associate the events raised by the XmlValidatingReader object with your event-handling procedure. Listing 10.39 shows an example of this.

Listing 10.39 Responding to Validation Events Raised by the Validate Subroutine

```
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Schema" %>
<SCRIPT runat='server'>

    void Page_Load(Object Sender,EventArgs e)
    {
      XmlTextReader tr = new XmlTextReader(Server.MapPath("book-invalid.xml"));
      XmlValidatingReader vr = new XmlValidatingReader(tr);

      vr.ValidationType = ValidationType.Schema;
      vr.ValidationEventHandler += new ValidationEventHandler(ValidationHandler);
      vr.Schemas.Add(null, Server.MapPath("book.xsd"));

      while(vr.Read())
      {
        Response.Write("[" + vr.Name + "]" + vr.Value + "<BR>");
        if(vr.NodeType == XmlNodeType.Element)
        {
           while(vr.MoveToNextAttribute())
             Response.Write("[" + vr.Name + "]" + vr.Value + "<BR>");
        }
      }
    }

    public void ValidationHandler(Object sender, ValidationEventArgs args)
    {
      Response.Write("<P><B>Validation error</B><BR>");
      Response.Write("Severity: " + args.Severity + "<BR>");
      Response.Write("Message: " + args.Message + "<BR>");
    }

</SCRIPT>
```

This document is created with the unregistered version of CHM2PDF Pilot

You can see that the validation-handling procedure is a standard event handler assigned to the XmlValidatingReader with a call to the AddHandler statement. When a validation-handling procedure is assigned to the XmlValidatingReader in this way, the reader will issue calls to the validation-handling procedure whenever it encounters a validation error in the document.

NOTE

Because the ValidationEventHandler object is a member of the System.Xml.Schema object, you should import this namespace at the beginning of any code that uses the XmlValidatingReader object. To do this, use this page directive:

```
<%@ Import Namespace="System.Xml.Schema" %>
```

You can test your validation code by changing the XML document that the page parses. Do this by altering the constructor of the XmlTextReader object in the code: book.xml should be valid, whereas book-invalid.xml will cause the validation event handler to be triggered. (Remember in our schema definition earlier in this chapter, we defined an ISBN data type to be an alphanumeric string of no more than 10 characters.)

Creating XSD Schemas in Visual Studio .NET

You can use Visual Studio .NET to create XSD schemas, often without writing code. Visual Studio .NET provides a visual drag-and-drop interface for creating schemas; it supports IntelliSense and instant syntax checking, as you'd expect with any other kind of code you would write in Visual Studio.

To create an XSD schema in Visual Studio .NET, begin by creating a Web application project. Next, add an XSD file to the project by right-clicking the project in the Solution Explorer, and then selecting Add, Add New Item from the pop-up menu. Finally, from the Add New Item dialog box, choose XSD Schema.

The XSD schema designer looks similar to the other server-side designers in Visual Studio .NET—it's a blank page. At the bottom of the page are two tabs, labeled Schema and XML. These tabs enable you to easily switch back and forth between visual and code views of the schema; you can create the schema either by dragging and dropping schema definition objects onto the page or by editing the code directly.

You can add a definition to the schema visually in one of two ways: by right-clicking the page, selecting Add from the pop-up menu, and choosing a schema member, or by choosing a schema member from the toolbox. (The Visual Studio .NET toolbox has a whole section devoted to the elements, attributes, simple and complex types, and other members of an XML schema definition.)

Editing Schema-Validated XML Files in Visual Studio .NET

Visual Studio .NET enables you to edit XML files with the same color coding and syntax checking you'd expect from any other kind of code you edit in Visual Studio. If you use Visual Studio .NET to edit an XML file that is defined by an XSD schema, you gain a bonus benefit—IntelliSense support. This means that for an XML document that is

This document is created with the unregistered version of CHM2PDF Pilot

defined by an XSL schema, Visual Studio .NET will provide drop-down lists of valid elements and attributes as you edit the XML document.

Creating Schemas from Data Sources Using Visual Studio .NET

Visual Studio .NET has the capability to create XML schemas automatically from a data source. This means that you can set up a database and Visual Studio will reverse engineer the structure of the database into XML schemas.

Because this function is tightly coupled to VS .NET's data-access features, we'll cover it in Chapter 11, "Creating Database Applications with ADO.NET."

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Processing XML Documents Using Style Sheets

In the previous sections, you saw how to create a schema to define and validate the structure of an XML document. One of the reasons to use schemas is to make your applications more flexible and easier to maintain by dissociating the definition of an XML document from the data found in that document. XML also provides a technology that enables you to define how a given document should be displayed, in a way that is similarly dissociated from the data document itself. This technology is known as XML style sheets, or XSL.

Like XML schemas, XSL style sheets are themselves XML documents, so you don't have to learn a whole new set of data structures to use XSL. However, you will have to ascend a bit of a learning curve as you come to understand the tags and attribute settings offered by XSL.

In addition to defining rules for how XML documents are displayed and formatted, style sheets also perform another interesting and useful function. They can define rules for transforming one type of XML document into another. This process is known as a transformation. The subset of the style sheet language that is responsible for transformations is known as XSLT.

The capability of style sheets to define the display of an XML document and to perform transformations are somewhat different, so we'll cover them separately in this section, starting with transformations.

NOTE

Like the other XML technologies covered in this chapter, we won't cover every aspect of XSL here, only the basics and techniques for integrating XSL into ASP.NET applications using the XML-handling objects found in the .NET framework. For more information on XSL, see the W3C site for XSL located at http://www.w3.org/Style/XSL/. A link to the current (as of this writing) recommendation on XSLT is located at http://www.w3.org/TR/xslt.

Transforming XML Documents Using Style Sheets

No matter what kind of data you have, eventually it will need to be converted for use in some other context. This is the case even with a "universal" data format such as XML. Just because the data is in XML format doesn't mean that every XML-aware software process will be able to consume that data meaningfully. In many cases the document you've defined will need to be changed, or transformed, so that it can be consumed by some other program. This is where the XML Style Sheet Transformations (XSLT) come in.

XSLT is based on the idea that a set of rules (known as a style sheet) can be applied to an XML document (known as the source tree) and produce another XML document (known as the result tree).

This document is created with the unregistered version of CHM2PDF Pilot

For example, suppose you're a bookseller running a point-of-sale software application that exports information about books in a form similar to that shown in Listing 10.40.

Listing 10.40 Bookseller's Hypothetical Output

```
<?xml version="1.0"?>
<STOCK>
  <BOOK isbn="0805062971" ti="Fight Club" au="Chuck Palahniuk" />
  <BOOK isbn="0751320005" ti="Eyewitness Travel Guides: London" />
</STOCK>
```

This system might export a list of books sold on a daily basis. This list could be sent to one or more publishers' fulfillment systems, which would then automatically restock your shelves with fresh copies of these books to sell to your clamoring customers.

The tricky part here lies in getting your point-of-sale system to talk to the publisher's automated order system. The publisher might expose an XML schema that lets you know exactly what structure an inbound XML document needs to have to be valid, but how do you get the XML emitted by your point-of-sale system to arrange itself so that it's acceptable to the publisher's order system?

To answer this question, let's start by illustrating what a book order might look like in the publisher's hypothetical order system. Listing 10.41 shows such a document.

Listing 10.41 Publisher's Hypothetical Input

```
<?xml version="1.0"?>
<ORDER custid='10459' date='2001-03-19'>
  <BOOK isbn="0805062971">
    <TITLE>Fight Club</TITLE>
  </BOOK>
  <BOOK isbn="0751320005">
    <TITLE>Eyewitness Travel Guides: London"</TITLE>
  </BOOK>
</ORDER>
```

As you can see, most of the information provided by the point-of-sale system is used in the publisher's ordering system, but the structure of the document is different. First, the publisher's system requires that you add your customer ID and order date as an attribute of the ORDER node. Next, the title of the book is expressed as a child node of the BOOK node rather than as an attribute. Finally, the author name is discarded (because the publisher can look up the author name, if needed, given the ISBN number that uniquely identifies all books).

It's true that you could write complicated parsing code (possibly using XML DOM objects or other XML-handling objects found in the .NET framework) that transforms the structure of your data document to the structure required by the publisher's system. But such code would likely be inefficient and difficult to maintain. If the publisher's requirements changed at some point in the future, you'd have to go in and hack your export code so that your exported data would continue to conform to their requirements. Instead, transforming the data using XSLT means that you have a much smaller body of code to contend with.

To get you started, we'll include a few minimal examples of XSLT transformations in this chapter and give you

This document is created with the unregistered version of CHM2PDF Pilot

pointers on how to perform XSLT transformations programmatically using the XML-handling objects found in the .NET framework classes.

First, consider a document called stock.xml that contains output from the hypothetical point-of-sale application we mentioned earlier. To transform this document into a structure you can use, you first create an XSLT style sheet and then associate that sheet with the document.

Creating a Style Sheet to Transform Data

You use the XslTransform class to transform an XML document from one format to another. Transformations done using XSLT can convert XML files from one XML structure to another or convert an XML file to a different format entirely (such as HTML).

NOTE

The XslTransform class is found in the System.Xml.Xsl namespace. A reference to the classes, properties, and methods introduced in this chapter is included at the end of this chapter.

To transform an XML document, you first create an XSLT style sheet that determines how the input document will be transformed. You then associate the style sheet with the input document (typically using the Load method of the XslTransform object).

Creating an XSLT Style Sheet

An XSLT style sheet is an XML document that specifies how to convert, or transform, another XML document. As with the other XML applications we've discussed in this chapter, such as XSD schemas and XPath queries, XSLT has its own syntax but is ultimately built on an XML framework.

XSLT style sheets typically begin with a stylesheet declaration and a matching expression. The matching expression is an XPath query that determines which elements of the source document you want to convert. If you want to convert the entire source document, you use the expression match="/".

After you've determined the elements you want to convert, you then create a template to perform the transformation. This is similar to template-based programming in ASP.NET; you hard-code the way you want your output to appear and then include XSL expressions that control how the data is merged with the template.

Listing 10.42 shows an example of a simple style sheet that takes the books.xml data file and transforms it into an HTML table.

Listing 10.42 An XSLT Style Sheet That Converts an XML Document into an HTML Document

```
<?xml version="1.0" encoding="UTF-8" ?>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- 'match' attribute is an XPath expression -->
<xsl:template match="/">
 <HTML>
 <BODY>
     <TABLE border="1" cellspacing="0" cellpadding="3">
       <TR>
        <TD bgcolor='#CCCCCC'><B>Title</B></TD>
        <TD bgcolor='#CCCCCC'><B>Author</B></TD>
      </TR>
      <xsl:for-each select="BOOKS/BOOK">
      <TR>
        <TD>
                <xsl:value-of select="TITLE" />
        </TD>
        <TD>
                <xsl:value-of select="AUTHOR" />
        </TD>
      </TR>
      </xsl:for-each>
      </TABLE>
 </BODY>
 </HTML>
</xsl:template>
</xsl:stylesheet>
```

You can see the <xsl:stylesheet> heading at the beginning of the style sheet, as well as the <xsl:template> element that marks the beginning of the template. In the body of the template are HTML markup tags, looping constructs (for-each), and functions that extract the value from XML nodes and insert them into the output of the transformation (value-of). There are a number of other operations similar to for-each and value-of in XSLT; however, you can perform the vast majority of XSLT transformations with these two links.

To see the effect this style sheet has on the XML document, you can create a link from the XML file to the style sheet (described in the next section) and then view the XML file in Internet Explorer.

Associating a Document with a Style Sheet

To link an XML style sheet to an XML document, you use a processing instruction that is similar in syntax to the optional XML declaration. Listing 10.43 shows an example of how to link an XML file to an XSL style sheet.

Listing 10.43 Version of the books.xml File Containing a Link to the books.xsl Style Sheet

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="books.xsl"?>
<BOOKS>
  <BOOK>
    <TITLE>C# Developer's Guide To ASP.NET, XML and ADO.NET</TITLE>
    <AUTHOR id="101" location="San Francisco">Jeffrey P. McManus</AUTHOR>
    <AUTHOR id="107" location="Seattle">Chris Kinsman</AUTHOR>
  </BOOK>
  <BOOK>
    <TITLE>How to Pluck a Purdue Chicken</TITLE>
    <AUTHOR id="107" location="Seattle">Chris Kinsman</AUTHOR>
  </BOOK>
  <BOOK>
    <TITLE>My Life Among the Baboons</TITLE>
    <AUTHOR id="107" location="Seattle">Chris Kinsman</AUTHOR>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  </BOOK>
</BOOKS>
```

When you view this file in Internet Explorer 5.0 or later, you should be able to see the data formatted as an HTML table.

Note that if a hard-coded link to the style sheet exists in the XML document, you do not need to write ASP.NET code to perform the transformation—as long as you're viewing the XML file in Internet Explorer and the style sheet file is where the link element says it should be, the XML data should be transformed.

Note, too, that it is legal to include multiple xsl-style-sheet instructions in a given XML document. The XSL specification dictates that when multiple style sheets exist, they're interpreted one at a time, one after the other.

Programmatically transforming the document on the server using .NET code requires a bit more effort, however; this is discussed in the next section.

Performing XSL Transformations Programmatically

You may find yourself in a situation where you want to apply an XSL transformation to a given XML file, but you can't or don't want to hard-code a link to the style sheet in the XML document. Perhaps the file is streamed to your application via HTTP, or it resides in a file system in a read-only state. In this case, you must associate the XSL style sheet with the XML data programmatically.

After you have the data and the XSL style sheet, programmatically transforming an XML document is quite easy. To do this, you use the XslTransform object, first loading the XSL style sheet using the Load method and then transforming the XML data using the Transform method. Listing 10.44 shows an example of how this works.

Listing 10.44 Programmatically Transforming an XML File Using an XSL Style Sheet and the XslTransform Object

```
<%@ Page debug="true" ContentType="text/xml" %>
<%@ Import namespace="System.Xml" %>
<%@ Import namespace="System.Xml.XPath" %>
<%@ Import namespace="System.Xml.Xsl" %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    XslTransform xslt = new XslTransform();
    XPathDocument xpd = new XPathDocument(Server.MapPath("books.xml"));

    xslt.Load(Server.MapPath("books.xsl"));
    xslt.Transform(xpd, null, Response.OutputStream);

  }

</SCRIPT>
```

This document is created with the unregistered version of CHM2PDF Pilot

It just happens that we chose to stream the output of the transformation to the OutputStream of the Response object (thereby sending the data immediately to the browser). If you choose, you can use one of the many overloaded versions of the Transform method to stream the output to a file or to an XmlReader object for further processing.

Editing XSLT Files Using Visual Studio .NET

You can use Visual Studio .NET to create and edit XSLT documents. To do this, simply right-click a Web project, select Add from the pop-up menu, and choose Add New Item from the submenu. Then, from the dialog box, select XSLT File. Give the file a name and click the Open button, and a blank XSLT document will be created. Visual Studio .NET will create the boilerplate definition of your XSLT document, including a link to the W3C namespace for XSLT located at http://www.w3.org/1999/XSL/Transform.

You'll need to be careful with one thing when using Visual Studio .NET to create and edit XSLT style sheets—testing XSL by loading the XML document into Internet Explorer (as described in the section "Validating Documents Using W3C Schemas" earlier in this chapter) won't work with the XSLT boilerplate generated by Visual Studio .NET. This is because Internet Explorer versions 5.0 and 5.5 (the current version as of this writing) support an earlier revision of the XSL recommendation.

Specifically, Internet Explorer 5.x requires a link to the older version of the W3C namespace for XSL located at http://www.w3.org/TR/WD-xsl. Internet Explorer 6.0 works fine with the default XSLT schema supplied by Visual Studio .NET.

Fortunately, however, the XML-handling objects in the .NET framework support the current revision of XSLT, so you can use the XSLT editor in Visual Studio .NET to perform server-side transformations.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Class Reference

This section provides a quick interface reference to the key objects described in this chapter. Space constraints prevent us from documenting every object in the .NET framework in this book; for the sake of brevity and conciseness, we include only the most important objects here. For more information on the other objects in the .NET framework, consult the .NET Framework Reference online help file.

ValidationEventArgs Class

Member of System.Xml.Schema. Not inheritable.

A ValidationEventArgs object is passed by the ValidationEventHandler delegate used by the XmlValidatingReader object.

Properties		
Exception	Message	Severity
Methods		
Equals	GetHashCode	MemberwiseClone
Finalize	GetType	ToString

XmlDocument Class

Member of System.Xml.

The XmlDocument object represents the top-level object in the XML DOM hierarchy. You can use it to read and write any element of an XML document, as well as to retrieve an XML document through the local file system or over HTTP.

Properties		
Attributes	InnerXml	NodeType
BaseURI	IsReadOnly	OuterXml
ChildNodes	Item	OwnerDocument
DocumentElement	LastChild	ParentNode
DocumentType	LocalName	Prefix
FirstChild	Name	PreserveWhiteSpace
HasChildNodes	NamespaceURI	PreviousSibling
Implementation	NameTable	Value
InnerText	NextSibling	XmlResolver

This document is created with the unregistered version of CHM2PDF Pilot

Methods		
AppendChild	CreateTextNode	LoadXml
Clone	CreateWhitespace	MemberwiseClone
CloneNode	CreateXmlDeclaration	Normalize
CreateAttribute	Equals	PrependChild
CreateCDataSection	Finalize	ReadNode
CreateComment	GetElementById	RemoveAll
CreateDefaultAttribute	GetElementsByTagName	RemoveChild
CreateDocumentFragment	GetEnumerator	ReplaceChild
CreateDocumentType	GetHashCode	Save
CreateElement	GetNamespaceOfPrefix	SelectNodes
CreateEntityReference	GetPrefixOfNamespace	SelectSingleNode
CreateNavigator	GetType ImportNode	Supports ToString
CreateNode	InsertAfter	WriteContentTo
CreateProcessingInstruction	InsertBefore	WriteTo
CreateSignificantWhitespace	Load	

XmlNamedNodeMap Class

Member of System.Xml.

The XmlNodeList object is used to handle a collection of nodes. It is used in the XML DOM parser model.

Properties		
Count		
Methods		
Equals	GetNamedItem	RemoveNamedItem
Finalize	GetType	SetNamedItem
GetEnumerator	Item	ToString
GetHashCode	MemberwiseClone	

XmlNodeReader Class

Member of System.Xml.

Properties

This document is created with the unregistered version of CHM2PDF Pilot

AttributeCount	IsDefault	NodeType
BaseURI	IsEmptyElement	Prefix
CanResolveEntity	Item	QuoteChar
Depth	LocalName	ReadState
EOF	Name	Value
HasAttributes	NamespaceURI	XmlLang
HasValue	NameTable	XmlSpace
Methods		
Close	MoveToAttribute	ReadInnerXml
Equals	MoveToContent	ReadOuterXml
Finalize	MoveToElement	ReadStartElement
GetAttribute	MoveToFirstAttribute	ReadString
GetHashCode	MoveToNextAttribute	ResolveEntity
GetType	Read	Skip
IsStartElement	ReadAttributeValue	ToString
LookupNamespace	ReadElementString	
MemberwiseClone	ReadEndElement	

XmlNode Class

Member of System.Xml.

The XmlNode object represents a single node in the hierarchy of an XML document. It can contain a number of subordinate objects that represent attributes, other nodes, and so forth.

Properties		
Attributes	IsReadOnly	NodeType
BaseURI	Item	OuterXml
ChildNodes	LastChild	OwnerDocument
FirstChild	LocalName	ParentNode
HasChildNodes	Name	Prefix
InnerText	NamespaceURI	PreviousSibling
InnerXml	NextSibling	Value
Methods		
AppendChild	GetPrefixOfNamespace	ReplaceChild
Clone	GetType	SelectNodes
CloneNode	InsertAfter	SelectSingleNode
CreateNavigator	InsertBefore	Supports
Equals	MemberwiseClone	ToString
Finalize	Normalize	WriteContentTo
GetEnumerator	PrependChild	WriteTo
GetHashCode	RemoveAll	

This document is created with the unregistered version of CHM2PDF Pilot

GetNamespaceOfPrefix	RemoveChild	

XmlNodeList Class

Member of System.Xml.

The XmlNodeList object is a collection that enables you to iterate through a set of child nodes. It is used in the XML DOM parser model.

Properties		
Count	ItemOf	
Methods		
Equals	GetHashCode	MemberwiseClone
Finalize	GetType	ToString
GetEnumerator	Item	

XmlReader Class

Abstract class. Member of System.Xml.

Properties		
AttributeCount	IsDefault	NodeType
BaseURI	IsEmptyElement	Prefix
CanResolveEntity	Item	QuoteChar
Depth	LocalName	ReadState
EOF	Name	Value
HasAttributes	NamespaceURI	XmlLang
HasValue	NameTable	XmlSpace
Methods		
Close	LookupNamespace	ReadElementString
Equals	MemberwiseClone	ReadEndElement
Finalize	MoveToAttribute	ReadInnerXml
GetAttribute	MoveToContent	ReadOuterXml
GetHashCode	MoveToElement	ReadStartElement
GetType	MoveToFirstAttribute	ReadString
IsName	MoveToNextAttribute	ResolveEntity
IsNameToken	Read	Skip
IsStartElement	ReadAttributeValue	ToString

XmlNodeReader Class

This document is created with the unregistered version of CHM2PDF Pilot

Member of System.Xml.

The XmlNodeReader object enables you to navigate an XML document using a scrolling cursor model. It inherits from System.Xml.XmlReader, an abstract class.

Properties		
AttributeCount	IsDefault	NodeType
BaseURI	IsEmptyElement	Prefix
CanResolveEntity	Item	QuoteChar
Depth	LocalName	ReadState
EOF	Name	Value
HasAttributes	NamespaceURI	XmlLang
HasValue	NameTable	XmlSpace
Methods		
Close	MoveToAttribute	ReadInnerXml
Equals	MoveToContent	ReadOuterXml
Finalize	MoveToElement	ReadStartElement
GetAttribute	MoveToFirstAttribute	ReadString
GetHashCode	MoveToNextAttribute	ResolveEntity
GetType	Read	Skip
IsStartElement	ReadAttributeValue	ToString
LookupNamespace	ReadElementString	
MemberwiseClone	ReadEndElement	

XmlTextReader Class

Member of System.Xml. Inherits from System.Xml.XmlReader.

The XmlTextReader object provides read-only, forward-only access to an XML document.

Properties		
AttributeCount	Item	Prefix
BaseURI	LineNumber	QuoteChar
CanResolveEntity	LinePosition	ReadState
Depth	LocalName	Value
Encoding	Name	WhitespaceHandling
EOF	Namespaces	XmlLang
HasAttributes	NamespaceURI	XmlResolver
HasValue	NameTable	XmlSpace
IsDefault	NodeType	

This document is created with the unregistered version of CHM2PDF Pilot

IsEmptyElement	Normalization	
Methods		
Close	MoveToAttribute	ReadEndElement
Equals	MoveToContent	ReadInnerXml
Finalize	MoveToElement	ReadOuterXml
GetAttribute	MoveToFirstAttribute	ReadStartElement
GetHashCode	MoveToNextAttribute	ReadString
GetRemainder	Read	ResetState
GetType	ReadBase64	ResolveEntity
IsStartElement	ReadBinHex	Skip
LookupNamespace	ReadChars	ToString
MemberwiseClone	ReadElementString	

XmlTextWriter Class

Member of System.Xml. Inherits from System.Xml.XmlWriter.

The XmlTextWriter object enables you to write XML to a text stream (typically a text file).

The first constructor takes an instance of the TextWriter class or any object that inherits from it (including another XmlTextWriter object). The second form takes an instance of a System.IO.Stream object and an encoding value (one of the members of System.Text.Encoding). The third form takes a string that represents a filename and an encoding value.

Properties		
BaseStream	IndentChar	WriteState
Formatting	Namespaces	XmlLang
Indentation	QuoteChar	XmlSpace
Methods		
Close	WriteBinHex	WriteName
Equals	WriteCData	WriteNmToken
Finalize	WriteCharEntity	WriteNode
Flush	WriteChars	WriteProcessingInstruction
GetHashCode	WriteComment	WriteQualifiedName
GetType	WriteDocType	WriteRaw
LookupPrefix	WriteElementString	WriteStartAttribute
MemberwiseClone	WriteEndAttribute	WriteStartDocument
ToString	WriteEndDocument	WriteStartElement
WriteAttributes	WriteEndElement	WriteString
WriteAttributeString	WriteEntityRef	WriteSurrogateCharEntity
WriteBase64	WriteFullEndElement	WriteWhitespace

This document is created with the unregistered version of CHM2PDF Pilot

XmlValidatingReader Class

Member of System.Xml. Inherits from System.Xml.XmlReader.

The XmlValidatingReader object provides support for reading an XML file using validation. The object supports validation with the DTD, XDR, or W3C XML schema formats.

Properties		
AttributeCount	IsEmptyElement	Reader
BaseURI	Item	ReadState
CanResolveEntity	LocalName	Schemas
Depth	Name	SchemaType
Encoding	Namespaces	ValidationType
EntityHandling	NamespaceURI	Value
EOF	NameTable	XmlLang
HasAttributes	NodeType	XmlResolver
HasValue	Prefix	XmlSpace
IsDefault	QuoteChar	
Methods		
Close	MoveToAttribute	ReadInnerXml
Equals	MoveToContent	ReadOuterXml
Finalize	MoveToElement	ReadStartElement
GetAttribute	MoveToFirstAttribute	ReadString
GetHashCode	MoveToNextAttribute	ReadTypedValue
GetType	Read	ResolveEntity
IsStartElement	ReadAttributeValue	Skip
LookupNamespace	ReadElementString	ToString
MemberwiseClone	ReadEndElement	
Events		
ValidationEventHandler		

XmlWriter Class

Abstract class. Member of System.Xml.

Properties		
WriteState	XmlLang	XmlSpace
Methods		
Close	WriteBinHex	WriteName
Equals	WriteCData	WriteNmToken

This document is created with the unregistered version of CHM2PDF Pilot

Finalize	WriteCharEntity	WriteNode
Flush	WriteChars	WriteProcessingInstruction
GetHashCode	WriteComment	WriteQualifiedName
GetType	WriteDocType	WriteRaw
LookupPrefix	WriteElementString	WriteStartAttribute
MemberwiseClone	WriteEndAttribute	WriteStartDocument
ToString	WriteEndDocument	WriteStartElement
WriteAttributes	WriteEndElement	WriteString
WriteAttributeString	WriteEntityRef	WriteSurrogateCharEntity
WriteBase64	WriteFullEndElement	WriteWhit espace

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Chapter 11. Creating Database Applications with ADO.NET

IN THIS CHAPTER

- Why a New Object Library for Data Access?
- New Features in ADO.NET
- Connecting to a Database
- Running Queries
- Using Data Adapters to Retrieve and Manipulate Data
- Creating Web Forms for Data Entry
- Handling Errors
- ADO.NET Framework Reference

The advent of .NET has opened an opportunity for Microsoft to revisit the design and architecture of all its core APIs with an eye toward making them simpler and more logical. This chapter examines how you can use ADO.NET to perform the common types of database operations that every application needs to perform and how you can take advantage of the new features of the database API to create more effective data-driven applications.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Why a New Object Library for Data Access?

A database application programming interface (API) is a software library that programmers use to perform useful work with a database. Prior to Microsoft.NET, many Web developers on the Microsoft platform used a database API called ActiveX Data Objects (ADO), a COM library that enabled access to data sources.

A notoriously cranky lot, database programmers tend to be irked when the database access API they use is changed. This is because so much depends on the database layer operating correctly and efficiently. The situation is exacerbated when the current database API works well and the new API doesn't provide many compelling benefits.

The version of ADO used by application developers prior to Microsoft.NET worked well for building client/server and traditional Web applications. But the world of application development continues to evolve, and this evolution demands that the tools we use evolve with it. For example, you can manipulate a resultset in the absence of a dedicated database connection in both ADO.old and ADO.NET. In contrast, only ADO.NET provides support for a new type of resultset object (the DataSet), which is relatively easy to create, always disconnected, universally available across any data provider, and powerful.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

New Features in ADO.NET

Many of the new features in ADO.NET represent gently evolved ways to perform operations that ADO.old programmers have always done, but some features for accessing and presenting data are fairly abrupt departures from what you may be accustomed to. Here is a high-level summary of some of the most important data-access changes to an ASP.NET developer.

Support for Disconnected Access and Remoting

ADO.old provided support for sessionless access through the disconnected recordset. To disconnect from a data source in ADO.old, you had to remember to set a specific cursor type and perform a series of magic incantations to make the recordset disconnect from the server. And then, when it was all done, there wasn't any straight-forward way for the developer to determine whether the operation actually worked—maybe it was disconnected, maybe not. Disconnected recordsets in ADO.old were both difficult to test and lacked adequately detailed documentation.

ADO.NET, on the other hand, makes it much easier to retrieve and manipulate disconnected data, including both relational constructs (formed of rows and columns) and hierarchical ones (such as XML documents). It does this through the use of a dedicated, always-disconnected object, the DataSet object, which we'll discuss at length later in this chapter.

XML Support

Although it was possible to express data contained in the ADO.old Recordset object as XML, the process wasn't as straightforward as it might have been. Support for XML was added as an afterthought in ADO.old, and it was supported in an extremely clunky way; in fact, you could argue that ADO.old's XML support was so convoluted that it made sense to use it only in the context of data operability with other ADO data sources. This runs totally counter to the spirit of data interoperability using XML—if the data can't go anywhere, there's really no point in using XML to begin with!

XML support in ADO.NET is provided through the DataSet object. The DataSet can always be rendered as XML with a single method call, and the XML that it renders is structured in a way that's easy to work with, whether you're working within the .NET framework or on some other platform.

Factored Data Access API

Developers liked ADO.old because the number of objects it provided was small and easy to understand. In fact, it was possible to perform most common data operations with a limited knowledge of only two objects: the Recordset and Connection objects.

The architects of ADO.NET have attempted to divide its functionality in a more granular fashion. The objective behind this is to give you more lightweight objects. For example, if you don't need support for advanced features such as disconnected access and remoting via XML, you can use a DataReader object instead of the DataSet

This document is created with the unregistered version of CHM2PDF Pilot

object. One unfortunate side effect is that there are more objects to learn. But the real advantage of a well-factored API is better performance and, in the long run, a simpler API. Now you don't have to internalize all the advanced features of ADO.NET until you're ready to use them.

No Support for Server-Side Cursors

A cursor is a software construct that enables your code to step though rows in a resultset one row at a time. There are different types of cursors; some cursors enable you to jump around from one record to another in the resultset with impunity (so-called scrolling cursors). In the Microsoft database cosmology, there are also "forward-only" or "firehose" cursors, which permit you only to move from the beginning of the recordset to the end without moving backward; some database programming purists argue that a cursor that does not scroll is not really a cursor at all.

In addition to scrolling or forward-only, cursors can reside in memory on the server or on the client. (For the purposes of this discussion, a middle-tier machine or a machine that provides access to data via Web services can be considered a client, too. Basically, any machine that is not the database server can be considered a client in this respect.) A server-side cursor allows you to perform server processing on the server without marshaling more results than you need to the client. Because marshaling data over the network can require a lot of time, database programmers sometimes turn to server cursors to perform processing for certain types of operations on database platforms (such as Microsoft SQL Server) that support them.

The problem with server-side cursors is that they consume resources on the server and provide only single-row access; server-side cursors don't support batch cursors.

In the meantime, one workaround is to embed server-side cursor functionality in a stored procedure, and then call that stored procedure using the normal ADO.NET techniques for calling stored procedures (described later in this chapter). The drawback of this workaround is that you must write your stored procedure in whatever language your database server uses. (This language is Transact-SQL if you're using Microsoft SQL Server.)

Scenarios for Using ADO.NET

One of the most common questions developers ask when attacking a new API is how to do the most basic operations that they are accustomed to performing with the perfectly good API that they already use. Table 11.1 gives an example of some of the most common scenarios for database access and describes the approach you take to implement them in ADO.NET.

Table 11.1. Common Database-Access Scenarios with ADO.NET

Scenario	Steps to Perform
Retrieve read-only data from a database	Open a connection; create a command object; create a DataReader object
Retrieve data from a database in read/write mode	Open a connection; create a data adapter; assign commands for selecting, reading, writing, and updating to the data adapter; fill the DataSet

This document is created with the unregistered version of CHM2PDF Pilot

Display the results of a query in a Web Forms control	Open a connection; create a data adapter; create a DataSet object; fill the DataSet; bind the control to a DataView object contained by the DataSet
Manipulate query results as XML	Open a connection; create a data adapter; create a DataSet object; send XML from the DataSet to a XmlTextReader via a call to the DataSet's ReadXml method
Retrieve a single (scalar) value from a query or stored procedure	Open a connection; create a command object; call the ExecuteScalar method of the command object
Execute a stored procedure that inserts, updates, or deletes data	Open a connection; create a command object; call the ExecuteNonQuery method of the command object

The remainder of this chapter describes how to perform these operations in detail.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Connecting to a Database

In ADO.NET, as in many object-based data-access APIs, you use a connection object to establish a connection with a database. In the ADO.NET SQL Server provider, the object is called SqlConnection; in the OLE DB provider, it's cleverly called OleDbConnection. Both objects are conceptually identical, although their implementation details differ slightly. The main difference between the two objects has to do with the syntax of the connection string. The connection string is required to tell the object how to get to the database that you're interested in.

NOTE

The SqlConnection class is a member of the System.Data.SqlClient namespace. The OleDbConnection class is a member of System.Data.OleDb namespace. A full list of members of both namespaces can be found in the reference section at the end of this chapter.

To connect to a SQL Server database using the SqlConnection object, use code like that shown in Listing 11.1.

Listing 11.1 Opening a Connection Using the SQL Server Provider

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    cn.Open();
    Response.Write("Opened connection to " + cn.Database + "<BR>");
    Response.Write("SQL Server version " + cn.ServerVersion);
  }

</SCRIPT>
```

This code opens the connection and displays information about the connection status. (You may need to change the server, database, uid, or pwd parameters of the connection string in this code to match the configuration of your system.)

As in ADO.old, one of the most important steps to establishing a connection is providing adequate and accurate information in the connection string that you pass to the connection object. For SQL Server, this information almost invariably includes a server name, database name, user ID, and password; additional settings can appear in the connection string, depending on your configuration and which database you are using.

ADO.old veterans will note that the syntax of the ADO.NET connection string is identical to that of ADO.old's

This document is created with the unregistered version of CHM2PDF Pilot

Connection object; this is one of the few areas where the techniques you use of ADO.old and ADO.NET intersect.

To do the same thing using the OLE DB provider, use the code shown in Listing 11.2.

Listing 11.2 Opening a Connection Using the OLE DB Provider

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>

<% @Import namespace='System.Data.OleDb' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    OleDbConnection cn;
    cn = new OleDbConnection("provider=SQLOLEDB;server=localhost;" +
                "uid=sa;pwd=;database=pubs;");
    cn.Open();
    Response.Write("Opened connection to " + cn.Database + "<BR>");
    Response.Write("SQL Server version " + cn.ServerVersion);
  }

</SCRIPT>
```

This should produce the same result that the previous example produced.

This OLE DB version of the code differs only slightly from the SQL Server version of the code. It imports the System.Data.OleDb namespace instead of System.Data.SqlClient, and it uses an OleDbConnection object instead of a SqlConnection. The connection string is also different: The OleDbConnection object requires a provider= clause to specify the type of OLE DB data source you want to use.

The capability to specify different OLE DB providers in the connection string is a powerful feature; it means that you can use ADO.NET to gain access to OLE DB data sources today, even if the database vendors don't yet support Microsoft.NET with providers of their own. If you can get to a data source using ADO.old, you can get to it using the ADO.NET OLE DB provider.

After you're done with your connection object, you should close it using its Close method. This will free up the computational resources devoted to that connection. In addition, you should handle any exceptions that are thrown by the Open method whenever you attempt to open a database connection.

Listing 11.3 contains a more complete example of opening a connection to a data source in ADO.NET, including explicit use of the Open and Close methods and error handling. Note that the structure of the error handler is such that the Close method is executed only if the Open method is successful.

Listing 11.3 Opening a Database Connection with Error Handling

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    try
    {
      cn.Open();
      Response.Write("Opened connection to " + cn.Database + "<BR>");
      Response.Write("SQL Server version " + cn.ServerVersion);
      cn.Close();
    }
    catch(SqlException sx)
    {
      Response.Write("Connection failed: " + sx.Message);
    }
  }

</SCRIPT>
```

The easiest way to test this code is to do something to make your connection string invalid, such as change the server name to a server that doesn't exist, or get rid of the connection string entirely. If the connection works, you'll see the name of the database and version of the server. (Note that SQL Server 2000 is considered to be SQL Server version 8.0 for versioning purposes.) If it fails, you'll get the error message contained in the Message property of the SqlException object.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Running Queries

After you have a connection to a data source, you can retrieve data from it by executing a query. The results of a query are stored in one of the ADO.NET data objects, typically either a DataReader or a DataSet object. The type of object can differ, depending on how you intend to work with the data.

This is in contrast to ADO.old, which invariably returned data in the form of a Recordset object. In ADO.NET, several objects can contain data. Table 11.2 summarizes these objects.

Table 11.2. Summary of Data-Containing Objects in ADO.NET

Object	Description
DataSet	Read/write; connectionless; contains one or more DataTable objects with relationships defined in a collection of DataRelation objects; supports filtering and sorting; supports automatic serialization to XML; contains an object (the DataView) that can be bound to data-aware ASP.NET Web forms controls
DataReader	Read-only; connection-based; supports scrolling forward only
DataTable	Contained by the DataSet; contains sets of rows and columns
DataColumn	Contained by the DataSet object; defines the schema (structure) of a DataTable
DataRow	Contained by the DataSet object; stores the data in an individual row

As you can see, the DataSet object supports far more features than the DataReader. But as with all benefits in technology, there are trade-offs in terms of programming complexity and performance. The key is to understand when it's most appropriate to use the various options available to you; these issues will be discussed in the next few sections of this chapter.

About the SqlDataReader Object

You can use the SqlDataReader object to execute queries and retrieve data from a data source. You typically use the SqlDataReader object in situations where you want to get a small quantity of data and display it quickly. If you're interested in retrieving and manipulating larger sets of data or you want to perform actions such as updating and deleting records, you will need to use a data command object or the DataSet object, discussed later in this chapter.

NOTE

The SqlDataReader class discussed in this section is a member of the System.Data.SqlClient namespace. The OLE DB provider equivalent of this class is called OleDbDataReader; it is found in the System.Data.OleDb namespace. A

This document is created with the unregistered version of CHM2PDF Pilot

full list of members of the SqlDataReader class can be found in the reference section at the end of this chapter.

The SqlDataReader is the rough equivalent of the read-only, forward-only Recordset object found in ADO.old. To create a SqlDataReader, you begin by creating a connection to the database and then executing a command that returns data. Using command objects to execute commands against data sources is discussed in the next section.

Executing Commands Using Command Objects

ADO.old provides the ADO Command object to execute commands against data sources. The ADO Command object can execute queries that retrieve data; it can also execute commands that insert, update, or delete data (these commands are referred to as data-manipulation commands). Finally, command objects can also execute stored procedures, which are bits of data-access code stored on the server.

ADO.NET, in contrast, gives you two ways to execute commands against data sources: the command object and the data adapter. The ADO.NET Command object is conceptually similar to the ADO.old Command object; it's used when you want to execute a quick command against a data source. The ADO.NET data adapter, in contrast, is used when you need to perform more involved operations against the back end—those operations that might have required a cursor in ADO.old.

Data adapters are also useful when you need to marshal a complex or hierarchical set of data remotely. Because they are based on XML, you could (in theory) remote the DataSet to another platform (one that does not contain a .NET implementation), manipulate the data—including making inserts and updates—and then remote the DataSet back to the .NET server. Putting this to work in practice would pose challenges (it would depend on the quality of tools available on the non-.NET platform), but the promise of standard XML makes it possible.

Ultimately, think of a command object as a way to manipulate data on the server or to return a data reader. The data adapter, in contrast, is the way to create a DataSet object, which establishes communication between the DataSet object and a specific data source.

In this chapter, we'll start simply, using the data reader and command objects first, and then move on to the more complex and feature-rich data adapter and DataSet objects.

Executing Text-Based Select Commands Using a Data Reader Object

A text-based command is a command that is constructed in your code at runtime. Commands are expressed in a query language that your data source can understand; this is typically (although not necessarily) Structured Query Language, or SQL.

Text-based commands are in contrast to stored procedures, which are defined ahead of time and reside on the server. (We'll discuss how to call stored procedures later in this chapter.) Text-based commands can perform selects (commands that retrieve data) or data manipulation (commands that insert, update, or delete data).

This document is created with the unregistered version of CHM2PDF Pilot

You can use a text command to retrieve a data reader object. Listing 11.4 shows an example of a text-based command that executes a select query (using a SqlCommand object) and returns a SqlDataReader object.

Listing 11.4 Executing a Text Command That Returns a SqlDataReader Object

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    SqlConnection cn;
    SqlCommand cm;
    SqlDataReader dr;
    String strSQL;

    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    strSQL = "SELECT TOP 10 au_fname, au_lname FROM authors";
    cm = new SqlCommand(strSQL, cn);

    // ** Open connection
    try
    {
      cn.Open();
    }
    catch(SqlException sx)
    {
      Response.Write("Connection failed: " + sx.Message);
    }

    // ** Execute command
    dr = cm.ExecuteReader();

    while(dr.Read())
    {
        Response.Write(dr["au_fname"] + " " + dr["au_lname"] + "<BR>");
    }

    cn.Close();

  }
</SCRIPT>
```

This code example builds on the ADO.NET connection object examples used in Listing 11.1. It adds a SqlCommand object that retrieves the first and last names of the first 10 authors in the pubs database. It also includes a SqlDataReader object that provides access to the information retrieved by the query.

NOTE

The SqlCommand class discussed in this section is a member of the System.Data.SqlClient namespace. The OLE DB provider equivalent of this class is called OleDbCommand; it is found in the System.Data.OleDb namespace. A full list of members of the SqlCommand class can be found in the reference section at the end of this chapter.

This document is created with the unregistered version of CHM2PDF Pilot

You can see from the code that the SqlDataReader object is initially created with a call to the ExecuteReader method of the SqlCommand object. If you're familiar with ADO.old, this pattern may be familiar to you—you can return an ADO Recordset object through a call to the Execute method of the ADO.old Command object as well.

After you have successfully created the SqlDataReader object, you can traverse the reader by executing its Read method in a loop. The Read method returns a true or false value depending on whether any data is readable; when you reach the end of data, the object returns false.

NOTE

The way that data reader objects in ADO.NET retrieve data can be contrasted to the somewhat awkward method that ADO.old employed to retrieve field values from a Recordset object—in ADO.old, you typically set up a while loop with a call to the MoveNext method inside the loop. If you forgot to include a call to the MoveNext method, your loop became infinite, and you kicked yourself for making a silly mistake. (You may have guessed that one of the authors of this book commits this error more frequently than he would like.)

As you work with data reader objects, remember that the data they store is read-only and forward-only. You can't make changes to the data returned by the data reader object (at least not through the data reader itself), and you can't jump around in the resultset like you can with a cursor-based resultset. However, the data reader is among the best-performing methods of all the data access methods in ADO.NET. As a result, data reader objects are useful in situations where you want to display or export data quickly without complex code or repeated interaction with the data source.

Executing Stored Procedures That Return Data

A stored procedure is a small chunk of code that is embedded in a database server. Stored procedures are commonly used in database applications because they execute more efficiently than database commands constructed on-the-fly. They also serve to separate data access code from business logic, which can be helpful for managing change and complexity in a database application.

NOTE

This section describes how to call stored procedures, but does not cover how to create them (that would require another whole book). Fortunately, such books have already been written. Two good books on writing stored procedures in SQL Server are *Transact-SQL Programming* (O'Reilly) and *The Guru's Guide to Transact-SQL* (Addison-Wesley). However, note that as of this writing, neither of these books covers recent changes in Transact-SQL introduced in SQL Server 2000. They're still great for learning both basic and advanced T-SQL programming and contain examples that demonstrate at least 90% of the operations you're ever likely to perform as a T-SQL programmer.

You can call stored procedures from ADO.NET using an ADO.NET command object (OleDbCommand in the

This document is created with the unregistered version of CHM2PDF Pilot

OLE DB provider or SqlCommand in the SQL Server provider). You can also specify a stored procedure as one of the four commands in a data adapter object. The four commands that are possible as data adapter object properties are SelectCommand, UpdateCommand, InsertCommand, and DeleteCommand.

Most stored procedures take parameters; for example, a stored procedure that performs a reporting and aggregation function might take a begin date and an end date as parameters. Similarly, a procedure to return information about a given customer or a given order would typically take a customer ID or order ID as one of its parameters. Parameterization enables you to limit the amount of data returned by the stored procedure.

We'll discuss stored procedures that take parameters later in this chapter. For now, we'll start simply and call a parameter-free stored procedure in the Northwind database. The name of this procedure is "Ten Most Expensive Products" and, not surprisingly, it is a SELECT procedure that should always return 10 rows of data.

In the SQL Server provider, the connection object is called SqlConnection; in the OLE DB provider it is called OleDbConnection. Both objects work essentially the same way, but for our examples in this section, we'll use the SQL Server command object. The most common SqlConnection constructor takes two arguments: a command in the form of a string and a connection object. Executing a command that selects records produces a SqlDataReader object that you can then use to display information.

Calling a stored procedure that returns data is similar to executing a text command that returns data. To call a stored procedure, you first create a connection and a command object, and then set the command object's CommandType property to the enumerated value CommandType.StoredProcedure.

Listing 11.5 shows an example of how to execute a query and retrieve data by calling a stored procedure using a SqlCommand and SqlDataReader object.

Listing 11.5 Calling a Simple Stored Procedure Using the SqlCommand Object

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    SqlCommand cm;
    SqlDataReader dr;

    cn = new SqlConnection("server=localhost;uid=sa;" +
                "pwd=;database=Northwind;");
    cm = new SqlCommand("Ten Most Expensive Products", cn);
    cm.CommandType = CommandType.StoredProcedure;

    // ** Open connection
    try
    {
      cn.Open();
    }
    catch(SqlException sx)
    {
```

This document is created with the unregistered version of CHM2PDF Pilot

```
      Response.Write("Connection failed: " + sx.Message);
    }

    // ** Execute command
    dr = cm.ExecuteReader();

    while(dr.Read())
    {
        Response.Write(dr.GetString(0) + " " + dr.GetDecimal(1) + "<BR>");
    }
    dr.Close();
    cn.Close();

 }

</SCRIPT>
```

This code again modifies Listing 11.4, which demonstrates executing a text command and retrieving a SqlDataReader object. In addition to setting the CommandType property of the SqlCommand object to CommandType.StoredProcedure, we also replace the command text with the name of the stored procedure ("Ten Most Expensive Products"). This user-defined stored procedure returns two columns of data: the name of the product and a column called UnitPrice. The data type of the UnitPrice column is of the currency data type (called "money" in SQL Server parlance) but is expressed as a decimal data type in .NET because that's how .NET deals with currency data. As a result, we use the GetDecimal method of the SqlDataReader object to extract prices from each row.

Passing Parameters to Stored Procedures

Just like function calls and subroutines in your code, most stored procedures take parameters. When you call a stored procedure that has one or more parameters, you must supply values for those parameters in code before you execute the stored procedure.

In ADO.NET, you can pass parameters to stored procedures in two ways. To pass parameter values, you can either include the parameter as part of the command itself (this is known as the inline method of passing parameters), or you can use the Parameters collection of the connection object. (This technique is syntactically closer to the ADO.old method of passing stored procedure parameters, but requires a bit more code than the inline approach.)

Like the previous examples, calling a command requires a connection object and a command object that are appropriate to the provider you're using. As we've been doing throughout this chapter, we'll stick with the SQL Server objects, SqlConnection and SqlCommand.

For our stored procedure, we'll use the SalesByCategory stored procedure in the Northwind database. This is a fairly simple select procedure that requires a single parameter: a product category (a string value found in the list of product categories in the Categories table), examples of which include Beverages, Produce, and Seafood.

Listing 11.6 shows how to pass a parameter to a stored procedure using the Parameters collection contained by the Command object.

Listing 11.6 Calling a Parameterized Stored Procedure Using the Parameters Collection of the

This document is created with the unregistered version of CHM2PDF Pilot

SqlCommand Object

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    SqlCommand cm;
    SqlDataReader dr;
    SqlParameter sp;

    cn = new SqlConnection("server=localhost;uid=sa;" +
                    "pwd=;database=Northwind;");
    cm = new SqlCommand();
    cm.Connection = cn;
    cm.CommandType = CommandType.StoredProcedure;
    cm.CommandText = "SalesByCategory";

    // ** Add parameter and parameter value
    sp = cm.Parameters.Add(new SqlParameter("@CategoryName",
                            SqlDbType.NVarChar, 15));
    sp.Value = "Condiments";

    // ** Open connection
    try
    {
      cn.Open();
    }
    catch(SqlException sx)
    {
      Response.Write("Connection failed: " + sx.Message);
    }

     // ** Execute command
     dr = cm.ExecuteReader();

     Response.Write("<B>" + sp.Value + "</B><BR>");
     while(dr.Read())
    {
      Response.Write(dr.GetString(0) + " " + dr.GetDecimal(1) + "<BR>");
    }
    dr.Close();
    cn.Close();

  }

</SCRIPT>
```

This version of the code uses the Add method of the Parameters collection contained by the SqlCommand object, cm. By adding this object to the command object's Parameters collection and then setting its Value property to the value you want to supply to the parameter, you tell the stored procedure which category of data to retrieve.

Note that the Add method used in the example takes a SqlParameter object as an argument. You can use several other constructors for the SqlParameter, depending on how much information you want to specify about the parameter. However, in most cases, the constructor we used (specifying the parameter name, data type, and size) will be the one you use most frequently. Note that the values for SQL Server data types are found in the enumeration

This document is created with the unregistered version of CHM2PDF Pilot

System.Data.SqlClient.SqlDbType. All the SQL Server 2000 data types are found in this enumeration (including SqlDbType.BigInt, the 64-bit integer, and the Unicode types NChar, NText, and NVarChar).

With SQL Server, it's actually not necessary to use the Parameters collection to supply a parameter to a stored procedure. In fact, you can call any SQL Server stored procedure using the EXEC keyword. The syntax of EXEC is

```
EXEC procname [param1], [param2] ...
```

where procname is the name of the stored procedure. Parameters, if any, are passed in a comma-delimited list following the name of the stored procedure.

Listing 11.7 shows an example of using EXEC to call a parameterized stored procedure. As in the previous example, the resultset of the stored procedure is returned in the form of a SqlDataReader object.

Listing 11.7 Calling a Parameterized Stored Procedure

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    SqlCommand cm;
    SqlDataReader dr;

    cn = new SqlConnection("server=localhost;uid=sa;" +
                           "pwd=;database=Northwind;");
    cm = new SqlCommand();
    cm.Connection = cn;
    cm.CommandText = "EXEC SalesByCategory 'Beverages'";

    // ** Open connection
    try
    {
      cn.Open();
    }
    catch(SqlException sx)
    {
      Response.Write("Connection failed: " + sx.Message);
    }
    // ** Execute command
    dr = cm.ExecuteReader();

    while(dr.Read())
    {
     Response.Write(dr.GetString(0) + " " + dr.GetDecimal(1) + "<BR>");
    }

    cn.Close();
  }

</SCRIPT>
```

This document is created with the unregistered version of CHM2PDF Pilot

It's important to note here that an EXEC call is technically a text command that calls a stored procedure, rather than a "pure" stored procedure call (as in the previous code example). Because an EXEC call is a text command, we get rid of the line of code that specifies that the command is a stored procedure (by setting the CommandType property of the command object).

Also, when using EXEC to call a stored procedure, parameters must be passed in the order they're declared in the stored procedure itself (a potential deal-killer if you don't have access to the stored procedure source code). Also, make sure to delimit non-numeric values (including date/time values) with single quotes; numeric values don't need to be delimited.

Executing Commands That Do Not Return Data

ADO.NET command objects have a special method for calling commands that do not retrieve data. This is an ADO.NET concept that does not specifically exist in ADO.old; it presumably exists for performance reasons, because a procedure that does not return data can be managed slightly more economically than one that does.

Examples of commands that don't return data include most data manipulation commands (including inserting, deleting, and updating records) as well as certain types of administrative commands in SQL Server.

You call a stored procedure that does not return data using the ExecuteNonQuery method of the connection object. Listing 11.8 shows an example of this.

Listing 11.8 Executing a Nonquery Command

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    SqlCommand cm;
    SqlDataReader dr;
    String strSQL;

    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    strSQL = "INSERT INTO authors " +
             "(au_id, au_fname, au_lname, contract) " +
             "VALUES ('123-45-6789', 'Chris', 'Kinsman', 0)";
    cm = new SqlCommand(strSQL, cn);

    // ** Open connection
    try
    {
      cn.Open();
      // ** Execute command
      cm.ExecuteNonQuery();

    }
    catch(SqlException sx)
    {
      Response.Write(" Exception occurred: " + sx.Message);
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    }
    finally
    {
      if (cn.State == ConnectionState.Open)
        cn.Close();
    }


  }

</SCRIPT>
    try
    {
      // ** Open connection
      cn.Open();
      // ** Execute command
      cm.ExecuteNonQuery();
    }
    catch(SqlException sx)
    {
      Response.Write("Connection failed: " + sx.Message);
    }
    finally
    {
      if (cn.State == ConnectionState.Open)
        cn.Close();
}
```

Note that executing this code will not display anything in the browser; you'll need to run a query against the Authors table to determine whether the insert actually worked. Later in this chapter, we'll construct a user interface that enables you to easily do this.

This code inserts a new author into the Authors table in the pubs database. It starts by constructing a SQL INSERT command, storing it in the string variable strSQL. This command contains the minimum amount of information required to insert a new author (the contract and au_id fields require non-null values). Aside from the syntax of the SQL command itself and the call to the ExecuteNonQuery method of the SqlCommand object, this code is nearly identical to the examples of commands we've demonstrated earlier in this chapter.

Executing Stored Procedures That Return Scalar Values

Most queries return resultsets, which are similar to arrays comprising one or more rows and one or more columns. In ADO.old these were called Recordsets; in ADO.NET, results typically are stored in an object such as a data reader or DataTable.

It is possible, however, to run a query that returns a single value. Such values are known as scalars, and they can be retrieved using the ExecuteScalar method of the ADO.NET command object.

As with the ExecuteNonQuery method discussed in the previous section, the idea behind ExecuteScalar is to give you additional options for executing commands that presumably perform better than returning the results in the form of a data reader or other object.

This document is created with the unregistered version of CHM2PDF Pilot

NOTE

You'll notice that when describing ADO.NET methods such as ExecuteNonQuery and ExecuteScalar, we use the word presumably a lot when describing their performance benefits. Doing a scientific performance analysis of the relative performance benefits of these various methods isn't our intention and isn't really within the scope of this book. At any rate, you'll want to examine the various methods for retrieving data in real-world scenarios before committing to a particular technique.

Listing 11.9 shows an example of using the ExecuteScalar method to retrieve a single value from the Northwind database.

Listing 11.9 Using ExecuteScalar to Retrieve a Single Value from the Database

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>
<SCRIPT runat='server'>

  void Page_Load(Object Sender, EventArgs e)
  {
    SqlConnection cn;
    SqlCommand cm;
    SqlDataReader dr;
    String strSQL;

    cn = new SqlConnection("server=localhost;uid=sa;" +
            "pwd=;database=Northwind;");
    strSQL = "SELECT Count(CustomerID) FROM Customers";
    cm = new SqlCommand(strSQL, cn);

    // ** Open connection
    try
    {
      cn.Open();
    }
    catch(SqlException sx)
    {
      Response.Write("Connection failed: " + sx.Message);
    }

    //** Execute command
    Response.Write("The number of customers is: " + cm.ExecuteScalar());

   cn.Close();

 }

</SCRIPT>
```

This code prepares a SQL command that returns a single value and submits it to the server using the ExecuteScalar method of the SqlCommand object. The ExecuteScalar method returns whatever value was returned by the command.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Using Data Adapters to Retrieve and Manipulate Data

In database programming, it's common to retrieve data and then perform other actions on the data (such as inserting new data, updating existing data, and deleting data). Often an application will need to take different actions based on the results of one or more queries or stored procedure calls.

In ADO.old, this can be accomplished through the use of a cursor. A cursor-based approach is expensive, however, because it implies a persistent connection to the data source. ADO.old does support a batch-update model whereby the developer can retrieve data, perform updates, and then reconnect to the data source and commit the updates in one operation (known as a batch).

The .NET vision of scalable, disconnected data access demands a different approach, however. The .NET DataSet requires a sessionless, cursorless approach. To provide access to retrieving, inserting, updating, and deleting records in a single object without the use of cursors or persistent connections, ADO.NET provides the data adapter object.

Data adapters

-
- Are similar to command objects, except they can contain four separate commands (one each for select, insert, update, and delete).
-
-
- Can be used to create always-disconnected DataSet objects. DataSets can be serialized as XML.
-
-
- Decouple data-manipulation code from the data itself, making the data easier to remote and the data manipulation code easier to maintain and reuse.
-

Each ADO.NET provider has its own data adapter object. In the SQL Server provider, the data adapter is named SqlDataAdapter; not surprisingly, in the OLE DB provider, the data adapter class is called OleDbAdapter.

The implementation details of the data adapter may vary slightly from one provider to the next, but the basic purpose of the data adapter is the same across all providers: Data adapters provide a connectionless method to engage in rich interaction with data sources. By rich interaction, we are talking about operations that go beyond the simple requests and display of data. Data readers are described in the "About the SqlDataReader Object" section of this chapter.

Although each ADO.NET provider has its own data adapter, DataSet objects created by different adapters are the same; DataSets are totally interoperable across providers. This is an important aspect of ADO.NET's interoperability story; it provides a standard way to express relational or hierarchical data that can be manipulated in any language, on any platform.

This document is created with the unregistered version of CHM2PDF Pilot

The ultimate objective of a Web database application is to present data to the user and permit users to manipulate data in interesting ways in the browser. The next few sections will introduce data adapters and demonstrate the various things you can do with them in ASP.NET applications. To demonstrate the power and flexibility of the data adapter, we'll first need to take a detour and discuss the principles of building a database-driven user interface in ASP.NET.

Displaying Query Data in the Browser

Earlier code listings in this chapter gave several examples of displaying data in a Web browser using calls to Response.Write. This is basically the same way you send output data to the browser in ASP.old. However, with ADO.NET and ASP.NET Web Forms controls, you have new options that provide better structure and maintainability, as well as more powerful features.

One of these features is data binding. Data binding refers to the process of automatically mapping the fields in a database to the user interface. Performing data binding automatically is handy because it relieves the developer from having to write a large amount of tedious code associated with retrieving and displaying data.

The concept of data binding got a bad rap among Visual Basic developers for a number of valid reasons. Data binding promised the capability to create a rich database user interface with a minimum of coding. Simplicity was the objective.

But this simplicity came at a price. Thick-client data-bound applications typically ran slowly, consumed a persistent database connection resource whether they were doing any work or not, and were difficult for programmers to code against because much of the underlying data-access functionality was encapsulated in the form of an object (the Data control) that exposed a painfully limited set of properties, methods, and events associated with data operations. If you were interested in building a certain type of data browser application and performance wasn't an issue, data binding worked well. But if you needed to build anything more sophisticated than a simple data browser, binding to a Data control was problematic at best.

Data binding in ASP.NET is different from the data-control-centric, connection-based vision of thick-client VB.old. The problems involving a "no-code" solution aren't a problem in ASP.NET, because ASP.NET data binding doesn't use black-box abstractions like the VB.old Data control.

The next few sections discuss the objects and techniques involved in data binding in ASP.NET and give you some examples describing how to put data binding to work in your applications.

Creating a DataSet Object Using a Data Adapter

You can use a DataSet in conjunction with a data adapter object to retrieve data from a database in a manner similar to the DataReader example in Listing 11.4. Although you might not use this code to display data in this way in a real application, dumping the contents of a query into a DataSet and then into the browser is a useful stepping stone on the way to data binding with Web Forms controls (which we'll discuss next).

This document is created with the unregistered version of CHM2PDF Pilot

Listing 11.10 shows a very simple example of a database select query using the SqlDataAdapter and a DataSet object.

Listing 11.10 Using the SqlDataAdapter and the DataSet Object to Display Query Results

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    SqlConnection cn;
    SqlDataAdapter da;
    DataSet ds;
    String strSQL;

    strSQL = "SELECT TOP 10 au_fname, au_lname FROM authors";
    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    da = new SqlDataAdapter(strSQL, cn);
    // ** Fill DataSet
    ds = new DataSet();
    da.Fill(ds, "Authors");

    // ** Display data
    foreach(DataRow Author in ds.Tables["Authors"].Rows)
    {
      Response.Write(Author["au_fname"].ToString() + " " +
                     Author["au_lname"].ToString() + "<BR>");
    }

  }

</SCRIPT>
```

You can see in this example, the SqlDataAdapter object is created from a SqlConnection object; this is similar to the way we created the basic SqlConnection object earlier. To build a DataSet object from the SqlDataAdapter object, we declare and instantiate the DataSet and then pass the DataSet to the Fill method of the SqlDataAdapter. The connection to the database is implicitly opened and closed when you fill a DataSet in this way.

Executing the fill method executes the SELECT query; at this point, we can start accessing the data through the Rows collection contained by the (one and only) table contained by the DataSet.

This example is similar to some of the data reader examples from earlier in this chapter, but with more complexity and more code. Fear not; this is just the first example and doesn't scratch the surface of what the DataSet can accomplish.

One of the big differences you can see between the behavior of the DataSet and the data reader object is that the DataSet has a Tables collection that contains DataTable objects. The capability to contain multiple tables in a single object (potentially containing relationships defined by one or more DataRelation objects) is one of the defining characteristics of the DataSet objects. We'll take a look at the power of multiple tables in a single DataSet in Listing 11.14.

This document is created with the unregistered version of CHM2PDF Pilot

For now, it might be worthwhile to look at more efficient and structured ways to display data on the page. Outputting HTML to the browser using a loop containing calls to Response.Write works fine, but if you're interested in doing anything more complicated than displaying row-by-row data with line breaks, you will want a more powerful display technique. Fortunately, ASP.NET provides this in the form of data-bound Web Forms controls, which we'll discuss in the next section.

Binding a DataView Object to Web Forms Controls

You can display data in an ASP.NET Web Forms page by using data binding. To use data binding, you begin by executing a select command with a data adapter. This produces a DataSet object that contains a collection of DataTable objects; each DataTable contains a DataView object that can be connected to any Web Forms control capable of binding to data (including many HTML server controls).

To perform the actual data binding after the DataSet is created, you first set the Web Forms control's DataSource property to a DataView object contained by the DataSet, and then use the DataBind method of the ASP.NET Page object to initiate binding. This method, typically called in the Page object's Load event, serves to connect the user interface control(s) on the page with the DataSet object(s) you've created.

Listing 11.11 takes the code from the previous example and amends it to display its output in a Web Forms DataGrid control.

Listing 11.11 Displaying Query Data in a Web Forms DataGrid Control

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<HTML>
<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    SqlConnection cn;
    SqlDataAdapter da;
    DataSet ds;
    String strSQL;

    strSQL = "SELECT TOP 10 au_fname, au_lname FROM authors";
    cn = new SqlConnection("server=localhost;uid=sa;" +
            "pwd=;database=pubs;");
    da = new SqlDataAdapter(strSQL, cn);_

    // ** Fill DataSet
    ds = new DataSet();
    da.Fill(ds, "Authors");
    // ** Display data
    DataGrid1.DataSource = ds.Tables["Authors"].DefaultView;
    DataGrid1.DataBind();


  }

</SCRIPT>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
<BODY>

  <ASP:datagrid id='DataGrid1' runat='server' />


</BODY>
</HTML>
```

Nothing is really new here except for the changes in the "display data" section of the code. This time we simply assign the DefaultView property (a DataView object) of the Authors table (a DataTable object) to the DataSource property of the DataGrid control, which we named DataGrid1. You should be able to see that the ASP:datagrid definition in the HTML section of the code doesn't specify any property settings for the DataGrid control other than its ID, so all the defaults are in place. The result is a plain vanilla HTML table, outputted to the browser.

Binding Other Objects to Web Forms Controls

Web Forms controls can be bound to any object that implements the System.Collections.IEnumerable or System.Collections.ICollection interfaces. The DataView object supports the IEnumerable interface, which is why it can be bound to ASP.NET server controls. You can also build your own .NET classes that implement one of these interfaces if you're interested in binding custom classes to ASP.NET Web Forms controls.

Creating your own classes for data binding is beyond the scope of this book, but it is pretty easy to see how an existing class can be bound to an ASP.NET control. The ArrayList class (found in System.Collections) is a perfect candidate to use as a guinea pig to see how this works; ArrayList is simple to work with and implements both IEnumerable and ICollection.

Listing 11.12 shows an example of how to bind a DataGrid control to an ArrayList object.

Listing 11.12 Binding a DataGrid Control to an ArrayList Object

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Collections' %>

<HTML>
<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    ArrayList al = new ArrayList();
    al.Add("Alaska");
    al.Add("Alabama");
    al.Add("California");
    al.Add("Kentucky");

    // ** Display data
    DataGrid1.DataSource = al;
    DataGrid1.DataBind();
  }

</SCRIPT>

<BODY>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
  <ASP:datagrid id='DataGrid1' runat='server' />

</BODY>
</HTML>
```

The technique to bind the DataGrid control shown here is the same as the methods shown earlier in the chapter to bind to relational data; as far as the DataGrid control is concerned, no difference exists between different types of bindable objects. As long as the object supports IEnumerable or ICollection, it can be displayed through binding.

Note that data readers (the SqlDataReader and OleDbDataReader classes) support the IEnumerable interface and can therefore be bound to Web Forms controls as well.

Expressing a DataSet as XML

One of the advantages of retrieving data with a DataSet is that a DataSet can be accessed at the object level (through the collection of DataTable and DataRow objects contained by the DataSet) or on a raw XML level. The capability to process a DataSet as XML means that you can easily transfer a DataSet to other platforms that don't explicitly support Microsoft.NET or work with the DataSet with XML tools (whether or not they're explicitly built to support .NET).

You can use the GetXml method of the DataSet object to extract XML data from the result of a query. By setting the ContentType Page directive to text/xml, it's possible to see the XML output directly in the browser window.

As with many Web-based XML examples, Listing 11.13 works best when you use Internet Explorer 5.0 or later; Internet Explorer has support for parsing and displaying XML directly in the browser window.

Listing 11.13 shows an example of outputting a query in XML format to the browser window using this technique.

Listing 11.13 Sending Query Results to the Browser Window Using XML

```
<% @Page ContentType='text/xml' language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>

 void Page_Load(Object Sender,EventArgs e)
  {
    SqlConnection cn;
    SqlDataAdapter da;
    DataSet ds;
    String strSQL;

    strSQL = "SELECT TOP 10 au_fname, au_lname FROM authors";
    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    da = new SqlDataAdapter(strSQL, cn);

    // ** Fill DataSet
    ds = new DataSet();
    da.Fill(ds, "Authors");
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    // ** Display data
    Response.Write(ds.GetXml());

  }

</SCRIPT>
```

Instead of simply dumping XML to the browser window, you can also assign the XML output of a DataSet object to one of the XML-manipulating objects in the .NET framework. To do this, you pass an XmlDataReader object to the ReadXml method of the DataSet object. You can then manipulate the XmlDataReader object as you would any other XML document, including sending it to a file or passing it to another process.

NOTE

The XmlDataReader object is one of the objects that the .NET framework provides for reading XML. For more information on how the XmlDataReader works, see Chapter 10, "Using XML."

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Creating Web Forms for Data Entry

Now that you have a sense of how to retrieve and manipulate data using ADO.NET, you'll need to have a strategy for building user interfaces in ASP.NET to access database functionality.

Like other Web-programming paradigms, the process of inserting or updating data through a Web browser typically involves constructing an HTML form that contains an array of input controls. The user inserts or changes values in the controls on the form and then submits the form to the Web server. The server passes the form contents to a script that then forms the actual data operation.

Listing 11.14 shows a simple example of an ASP.NET page that facilitates data entry.

Listing 11.14 Simple Data-Entry Form

```
<% @Page language='c#' debug='true' trace='false' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<SCRIPT runat='server'>
  void Page_Load(Object Sender,EventArgs e)
  {
    if(Page.IsPostBack)
     {
      SqlConnection cn;
      SqlCommand cm;
      String strSQL;

      cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
       strSQL = "INSERT INTO authors " +
                "(au_id, au_fname, au_lname, contract) " +
                "VALUES ('" + txtID.Text + "', '" +
                txtFirstName.Text + "', '" +
                txtLastName.Text + "', '" +
                ChkToInt(chkContract) + "')";

      cm = new SqlCommand(strSQL, cn);


      // ** Open connection
      try
      {
        cn.Open();
        // ** Execute command
        cm.ExecuteNonQuery();
      }
      catch(SqlException sx)
      {
        Response.Write("Exception Occurred: " + sx.Message);
      }
      finally
      {
     if (cn.State == ConnectionState.Open)
         cn.Close();
      }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
      // ** Execute command
      Trace.Write("Command: " + cm.CommandText);

      // ** Clear form for next item
      txtID.Text = "";
      txtFirstName.Text = "";
      txtLastName.Text = "";
      chkContract.Checked = false;
    }
  }

  int ChkToInt(CheckBox chk)
  {
    if(chk.Checked)
      return 1;
    else
      return 0;
  }

</SCRIPT>

<html>
<head>
<title>ASP.NET Data Entry</title>
</head>

<body bgcolor="#FFFFFF" text="#000000">
<FORM runat='server'>
  <table width="300" border="0">
    <tr>
      <td>ID: </td>
      <td>
        <asp:textbox id="txtID" runat='server' />
      </td>
    </tr>
    <tr>
      <td>First Name: </td>
      <td>
        <asp:textbox id="txtFirstName" runat='server' />
      </td>
    </tr>
    <tr>
      <td>Last Name:</td>
      <td>
        <asp:textbox id="txtLastName" runat='server' />
      </td>
    </tr>
    <tr>
      <td>Contract:</td>
      <td>
        <asp:checkbox id="chkContract" runat='server' />
      </td>
    </tr>
  </table>
  <p>
    <asp:button id="btnSave" text="Save" runat='server' />
  </p>
</FORM>
</body>
</html>
```

This page takes the code used to perform a nonquery command (introduced earlier in this chapter) and attaches a simple user interface to it. Although it is minimal, the pattern set by this example, in Listing 11.8, forms the basis of a

This document is created with the unregistered version of CHM2PDF Pilot

great deal of Web-based data entry forms in ASP.NET.

The Page_Load event procedure performs the work involved in inserting the data into the database. Note that the data is sent to the database only if the IsPostBack property of the Page object is true. There's no point sending data to the database unless there's something to send. Note, too, that we explicitly cleared the contents of the controls on the form after inserting the data (this circumvents ASP.NET's default view state behavior).

The only other tricky thing on this page is the ChkToInt function. This function takes the Checked value returned by the CheckBox control and converts it into the 1 or 0 value required by SQL Server.

Two problems occur with this page—both related to validation. If the user enters an author with the same ID as an existing author, she will get an error. You can get around this problem by catching the SqlException that is thrown when the ExecuteNonQuery method is reached.

The next problem is less straightforward: how to ensure that the user actually enters valid data for the ID, first name, and last name fields. It happens that ASP.NET provides some powerful and flexible components for dealing with this problem. We'll cover them in the next section.

Performing Validation

Every software application should have code that ensures that data entered by users is valid. Web applications are no different.

In ASP.old, developers typically embedded validation logic in pages that also contained display logic; separating validation logic from presentation logic was tricky, but it was possible. If you were interested in performing a simple validation, such as making sure that a given text box contains a value between 1 and 10, the template-based design of ASP.old practically forced you to embed that validation code on the same page.

Incorporating client-side validation adds a new level of complexity to this problem. Because client-side validation is commonly done using JavaScript (for cross-browser compatibility), this forced you to embed code written in two very different languages in a single page. Madness!

ASP.NET solves this problem by creating language-neutral objects for performing validation. These come in the form of Web Forms server controls—validation objects that accept generic rules pertaining to the most common types of data validation. These objects are tightly integrated with the Page object so that when a Web form is submitted to the server, the validation object can communicate to the Page object that the field it validates has not passed the validation rule.

ASP.NET provides six validation objects:

-
- RequiredFieldValidator

This document is created with the unregistered version of CHM2PDF Pilot

- CompareValidator
- RangeValidator
- RegularExpressionValidator
- ValidationSummary
- CustomValidator

To perform validation, you create the Web form as you normally would and then attach validation controls to the input controls in the form. This is done by setting the validation control's ControlToValidate property. You can also validate a control against a variable or constant value instead by setting the validation's ValueToCompare property. You then assign validation parameters where appropriate. (Some validation controls, such as RequiredFieldValidator, don't need extra validation parameters.)

When the page is submitted to the server, the validation rule contained by the validation control is applied. Validation controls are invisible until their validation rules are violated; if the validation rule fails, an optional error message is displayed. Your code must provide a way to deal with the situation where a validation rule is violated. You can programmatically inspect the IsValid property of the Page object to quickly determine whether one or more validation controls were violated. For complex pages with many validation controls, you can also provide a list of all validation violations on the page; in fact, ASP.NET provides a special object (the ValidationSummary control) to perform this function.

Eight controls that can be associated with validation controls, shown next, ship with ASP.NET.

HTML controls:

- HtmlInputText
- HtmlTextArea

This document is created with the unregistered version of CHM2PDF Pilot

- HtmlSelect
-
-

- HtmlInputFile
-

Server controls:

-

- TextBox
-
-

- ListBox
-
-

- DropDownList
-
-

- RadioButtonList
-

To keep the examples simple, in this section we'll perform validation against the TextBox server control exclusively. Also, to make these examples briefer and easier to understand, we won't bother including the actual data access code in the validation examples here; we'll assume that you'll include code similar to that described earlier in this chapter to perform the actual database operation required to get your data into the database.

Validation controls work by generating DHTML and JavaScript for browsers that support them, and performing round trips to the server for browsers that do not. This greatly increases the efficiency of validation, ensuring that forms that contain bad data aren't sent across the network. The best part of this feature is that you don't have to learn JavaScript to make this client-side validation happen; validation controls can emit the appropriate JavaScript code automatically.

The next few sections describe how to use each validation control in more detail.

Validating for Required Fields

Required field validation forces the user to enter a value in the validated control. It's one of the easiest validators to implement because it doesn't require a separate property assignment to determine what the validation rule is. If a RequiredFieldValidator is attached to a control, the field is required and the page isn't valid unless the user puts something in the field.

Listing 11.15 shows an example of the RequiredFieldValidator control in action.

This document is created with the unregistered version of CHM2PDF Pilot

Listing 11.15 Requiring a Field Using the RequiredFieldValidator Control

```
<HTML>
<HEAD>
<SCRIPT language="C#" runat="server">

    void SaveBtn_Click(Object Sender,EventArgs e)
    {
       if(Page.IsValid)
          lblOutput.Text = "Record saved.";
       else
         lblOutput.Text = "";
    }

</SCRIPT>
</HEAD>

<BODY>
<FORM runat="server" ID="Form1">

<asp:textbox id=TextBox1 runat=server />

    <asp:RequiredFieldValidator id="RequiredFieldValidator2"
        ControlToValidate="TextBox1"
        Display="Static"
        Width="100%" runat=server>
        Please enter your name.
    </asp:RequiredFieldValidator>

    <asp:Button id=Button1 text="Save" OnClick="SaveBtn_Click" runat=server />
    <asp:label id='lblOutput' runat='server' />
</FORM>
</BODY>
</HTML>
```

As you can see from the code, this mode of validation is pretty straightforward—just specify the control you want to validate in the RequiredFieldValidator's ControlToValidate property, and then check to see if the page is valid in an event procedure attached to a control that submits the form (in this case, the button called Button1).

One of the main objectives of client validation is to catch bad input before your application performs expensive trips across the network. This example demonstrates how ASP.NET handles this; if you enter bad data, the page will not be submitted to the server.

You might find it illustrative to see how this validation is performed "under the hood" of the browser. To see the client code that is generated by the validation control, navigate to this page and create a validation error by clicking the button without typing anything into the text field. Watch the browser as the error is generated; you should notice that no progress bar is at the bottom of the window to indicate a jump across the network, and the page displays the error message instantly, without having to reload.

Next, use the View Source command to take a look at the HTML code generated by the ASP.NET page. As you scroll through the code, you should be able to see a reference to a JavaScript file called WebUIValidation.js. This file resides on the server (in a directory called \aspnet_client under the IIS root directory) but is downloaded and executed on the client side when a validation control is present in a Web form. The JavaScript function named

This document is created with the unregistered version of CHM2PDF Pilot

RequiredFieldValidatorEvaluateIsValid is called when you use a RequiredFieldValidator control (analogous functions exist for the other types of validators). By viewing the script file WebUIValidation.js, you can see how they work. The one for required field validation is monumentally trivial—it's listed in Listing 11.16.

Listing 11.16 Client-Side Validation for Required Field Function

```
function RequiredFieldValidatorEvaluateIsValid(val) {
    return (ValidatorTrim(ValidatorGetValue(val.controltovalidate)) !=
_ValidatorTrim(val.initialvalue))
}
```

This function uses a lot of verbiage to accomplish a simple task—figuring out whether a value is there. If it's there, the function returns true; if not, it returns false. Of course, this isn't so complicated that you couldn't have written it yourself, but it's nice that this kind of code is abstracted behind the validator control so that you don't have to think in two languages just to perform simple validation.

Comparison Validation Using the CompareValidator Control

Comparison validation examines the value of a control and compares it against the value of another control's property, a variable, or a constant. To use CompareValidator, you must specify three things: the control to validate, the control (or value) to compare it to, and an operator (one of the equality or inequality types).

As an example of this, suppose you're building a Web form that gives your employees pay raises. The important validation rule with a pay raise calculation is: Don't accidentally give your employees a pay cut! You can use a CompareValidator control with the greater-than operator to ensure that this is the case. Listing 11.17 shows an example.

Listing 11.17 Performing Comparison Validation in a Data Entry Form

```
<HTML>
<HEAD>
<TITLE>
Pay Raise Calculator
</TITLE>
<SCRIPT language="C#" runat="server">

    void SaveBtn_Click(Object Sender,EventArgs e)
    {
       if(Page.IsValid)
         lblOutput.Text = "The new pay rate is: " + txtNewRate.Text;
       else
         lblOutput.Text = "";
    }

</SCRIPT>
</HEAD>

<BODY>
<FORM runat="server">

Current Rate:<asp:textbox id=txtOldRate text='3000' runat=server /><BR>
New Rate:<asp:textbox id=txtNewRate runat=server />
<asp:Button id=Button1 text="Save" OnClick="SaveBtn_Click" runat=server /><BR>

    <asp:CompareValidator id="CompareValidator1"
```

This document is created with the unregistered version of CHM2PDF Pilot

```
        ControlToValidate="txtNewRate"
        ControlToCompare="txtOldRate"
        Type="Double"
        Operator="GreaterThan"
        runat="server">
    You eeeediot! Do not give your employees a pay cut!
    </asp:CompareValidator>

<asp:label id='lblOutput' runat='server' />
</FORM>
</BODY>
</HTML>
```

To understand the relationship between ControlToValidate, ControlToCompare, and Operator, think of the three properties as elements of an expression that looks like this:

```
ControlToCompare   Operator   ControlToValidate
```

Hence, if ControlToCompare is 3000, ControlToValidate is 3500, and Operator is "GreaterThan", the expression is true and the page is valid. If ControlToValidate is 0, for example, the expression becomes false and the validation fails.

The legal values for the Operator property for controls that use them are enumerated in System.Web.UI.WebControls.ValidationCompareOperator and are listed in Table 11.3.

Table 11.3. Members of the ValidationCompareOperator Enumeration Used in Comparison Validation

Member	Description
DataTypeCheck	Returns true if the two values are the same data type
Equal	Returns true if the two values are equal
GreaterThan	Returns true if ControlToValidate is greater than ControlToCompare
GreaterThanEqual	Returns true if ControlToValidate is greater than or equal to ControlToCompare
LessThan	Returns true if ControlToValidate is less than ControlToCompare
LessThanEqual	Returns true if ControlToValidate is less than or equal to ControlToCompare
NotEqual	Returns true if the two controls are not equal

The broad range of operators gives you a great deal of flexibility; however, if you need a validation rule that goes beyond what any of the standard validation controls are capable of, you can always turn to the custom validation control, as seen in Listing 11.20.

Range Validation Using the RangeValidator Object

Range validation forces the data in a given control to fall within a given range—alphabetic, numeric, or date. You specify the boundaries of the range using the control's MinimumValue and MaximumValue properties. As with the CompareValidator control, you can also denote a data type on which to base the comparison (using the control's

This document is created with the unregistered version of CHM2PDF Pilot

Type property).

Listing 11.18 shows an example of the RangeValidator object in action.

Listing 11.18 Performing Range Validation

```
<HTML>
<HEAD>
<TITLE>
Pay Raise Calculator [Range]
</TITLE>
<SCRIPT language="C#" runat="server">

    void SaveBtn_Click(Object Sender,EventArgs e)
    {
       if(Page.IsValid)
         lblOutput.Text = "The new pay rate is: " + txtNewRate.Text;
       else
         lblOutput.Text = "";
     }

</SCRIPT>
</HEAD>

<BODY>
<FORM runat="server">

Current Rate:<asp:textbox id=txtOldRate text='3000' runat=server /><BR>
New Rate:<asp:textbox id=txtNewRate runat=server />
<asp:Button id=Button1 text="Save" OnClick="SaveBtn_Click" runat=server /><BR>

    <asp:RangeValidator id="RangeValidator1"
         ControlToValidate="txtNewRate"
         MinimumValue='1000'
         MaximumValue='5000'
         Type="Double"
         runat="server">
    Please enter a value between 1000 and 5000.
    </asp:RangeValidator>
<asp:label id='lblOutput' runat='server' />
</FORM>
</BODY>
</HTML>
```

In this example, we revisit the pay raise calculator. This time, we want to make sure that our human resources executives aren't too generous or too stingy with our employees. To test this code, navigate to the page and attempt to enter a value less than 1000 or greater than 5000 as a new pay rate. The control will render the page invalid, preventing the data from being processed until you enter a valid amount.

You'll want to make sure that you always provide an error message that clearly informs the user what the valid range is when using a RangeValidator control. If you can, it's even better to let the user know what the valid range is initially, before the user has a chance to make an error.

Validation Using Regular Expressions

This document is created with the unregistered version of CHM2PDF Pilot

Regular expressions are a symbolic minilanguage used for text processing. You can use the RegularExpressionValidator control to apply a regular expression comparison to a value in your form. To do this, you assign the regular expression pattern to the ValidationExpression property of the RegularExpressionValidator control.

For example, suppose you're creating an application that requires users to establish a PIN number when they initially create their account. The rules for your application are that PINs must be composed of a single non-numeric character followed by three numbers. Determining whether a given string is four characters in length is easy in code, but determining whether those digits are numbers or letters may take a bit of doing. Fortunately, it's easy using a regular expression—the expression "\D\d\d\d" matches a string comprising a non-digit character followed by three digits. A comparison based on this expression will reject anything larger or smaller than four characters in length.

Listing 11.19 shows an example of how to perform regular expression validation in ASP.NET using this validation rule.

Listing 11.19 Performing Validation Based on a Regular Expression Comparison

```
<HTML>
<HEAD>
<TITLE>
Account Creation [RegExp]
</TITLE>
<SCRIPT language="C#" runat="server">

    void SaveBtn_Click(Object Sender,EventArgs e)
    {
       if(Page.IsValid)
         lblOutput.Text = "Account created!";
       else
         lblOutput.Text = "";
    }

</SCRIPT>
</HEAD>

<BODY>
<FORM runat="server" ID="Form1">

User ID:<asp:textbox id=txtUserID text='newuser' runat=server /><BR>
PIN:<asp:textbox id=txtPIN runat=server />
<asp:Button id=Button1 text="Save" OnClick="SaveBtn_Click" runat=server /><BR>

    <asp:RegularExpressionValidator id="RegularExpressionValidator1"
         ControlToValidate="txtPIN"
         ValidationExpression="\D\d\d\d"
         runat="server">
    Please enter PIN comprising a non-number followed by three numbers.
    </asp:RegularExpressionValidator>

<asp:label id='lblOutput' runat='server' />
</FORM>
</BODY>
</HTML>
```

To test this code, navigate to the page in a browser and attempt to enter an invalid PIN (such as XXXX) into the PIN field. You should be able to see that any combination of characters that does not match the regular expression will cause the validation rule to fail; a valid entry is a string similar to "A599."

This document is created with the unregistered version of CHM2PDF Pilot

NOTE

A full exploration of the syntax of regular expressions is beyond the scope of this book, but they're covered adequately in the .NET framework SDK documentation. The seminal book on regular expressions is Jeffrey E. F. Friedl's *Mastering Regular Expressions* (O'Reilly). The book is geared toward developers writing scripts using the Perl programming language in Unix, but don't let that scare you away—the amount of actual Perl code in the book is very small.

Custom Validation Using the CustomValidator Object

You can use custom validation in situations where your validation is too complicated or irregular for the standard validation controls described in this section.

To implement a custom validation rule using the CustomValidator control, you must write a custom validator function. The cool thing about CustomValidator is that your validation function can reside and execute on either the server side or the client side, or both. (Of course, if cross-browser compatibility is important to you, you will want to write your client-side validation function in JavaScript.) To assign a server-side validation function to a CustomValidator, you write a function and assign the name of the function to the ControlValidator's OnServerValidate property. To assign a client-side function, write a client function and assign it to the ControlValidator's ClientValidationFunction property.

To demonstrate this function, we'll return to the PIN example introduced in the previous example. This time, we'll assume that the user created an account and is ready to log in. We want to provide custom validation on the client and server side for the PIN field. On the client side, we want to make sure that the PIN is composed of a letter and three numbers; we will use a regular expression in JavaScript to do this. But you wouldn't want to verify the password using client-side JavaScript, because this technique isn't secure (JavaScript is executed on the client and is visible to the client). So we'll perform the actual verification of the PIN on the server.

Listing 11.20 shows an example of custom validation using server-side validation.

Listing 11.20 Performing Custom Validation on the Server Side

```
<HTML>
<HEAD>
<TITLE>
Login with PIN [Custom Validation]
</TITLE>

<SCRIPT language="C#" runat="server">

    void SaveBtn_Click(Object Sender,EventArgs e)
    {
       if(Page.IsValid)
         lblOutput.Text = "Login successful!";
       else
         lblOutput.Text = "";
    }
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    void ServerVerify(Object Sender, ServerValidateEventArgs Value)
    {
      // In a real application, this code would do
      // a database call or use Active Directory or
      // something interesting. For this example, no.
      if(txtPIN.Text == "A999")
        Value.IsValid = true;
      else
        Value.IsValid = false;
    }

</SCRIPT>
</HEAD>

<BODY>
<FORM runat="server">

User ID:<asp:textbox id=txtUserID text='myusername' runat=server /><BR>
PIN:<asp:textbox id=txtPIN runat=server />
<asp:Button id=Button1 text="Save" OnClick="SaveBtn_Click" runat=server /><BR>

    <asp:CustomValidator id="CustomValidator1"
         ControlToValidate="txtPIN"
         OnServerValidate="ServerVerify"
         runat="server">
    Invalid PIN number!
    </asp:CustomValidator>

<asp:label id='lblOutput' runat='server' />

</FORM>
</BODY>
</HTML>
```

You can see that the CustomValidator is pretty straightforward—to make it work, you set its OnServerValidate property to the name of the function you want to use to perform the validation. Your code then sets the IsValid property of the ServerValidateEventArgs object passed into the function to either true or false, depending on whether the validation logic contained in the function has validated the data.

To create a client-side validation function, you create a validation function and assign its name to the ClientValidationFunction property of the CustomValidator control. Your function should take two parameters: source and arguments (this is similar to an event procedure declaration). Within the client validation function, you set the value of arguments.IsValid to true or false, depending on the outcome of the validation code.

Also, you can write your client-side validation code in VBScript if cross-browser compatibility isn't important to you (that is, you know that every user who uses your Web application will be using some version of Microsoft Internet Explorer).

Note that if you don't want to perform client-side validation for some reason, you can simply omit it, as our code example has done in Listing 11.20.

Providing a Summary of Validation Rule Violations

This document is created with the unregistered version of CHM2PDF Pilot

Summary validation takes the error messages generated by any number of validation controls and displays them in a summary format. The ValidationSummary control can display its output inline on the page or in a pop-up message box. Because it has the capability to give the user multiple error messages at once, it's very effective in situations where you have many validation controls on a form.

Listing 11.21 shows an example of how summary validation works.

Listing 11.21 Displaying a Summary of Validation Errors

```
<HTML>
<HEAD>
<TITLE>
Pay Raise Calculator [Summary]
</TITLE>
<SCRIPT language="C#" runat="server">

    void SaveBtn_Click(Object Sender, EventArgs e)
    {
       if(Page.IsValid)
         lblOutput.Text = "The new pay rate is: " + txtNewRate.Text;
       else
         lblOutput.Text = "";
    }

</SCRIPT>
</HEAD>

<BODY>
<FORM runat="server" ID="Form1">

    <asp:RequiredFieldValidator id="RequiredFieldValidator1"
        ControlToValidate="txtName"
        ErrorMessage="Please enter an employee name."
        runat=server>
    *
    </asp:RequiredFieldValidator>

Employee Name:<asp:textbox id=txtName runat=server /><BR>
Current Rate:<asp:textbox id=txtOldRate text='3000' runat=server /><BR>

    <asp:RangeValidator id="RangeValidator1"
         ControlToValidate="txtNewRate"
         ErrorMessage="Please enter a value between 1000 and 5000."
         MinimumValue='1000'
         MaximumValue='5000'
         Type="Double"
         runat="server">*</asp:RangeValidator>

New Rate:<asp:textbox id=txtNewRate text='12.98' runat=server />

<asp:Button id=Button1 text="Save" OnClick="SaveBtn_Click" runat=server /><BR>

<asp:label id='lblOutput' runat='server' />

<asp:ValidationSummary id='ValidationSummary1' runat='server'
     DisplayMode='BulletList'
     HeaderText='You Committed Serious Data Validation Crimes:'
     ShowSummary='true'
     ShowMessageBox='true' />

</FORM>
```

This document is created with the unregistered version of CHM2PDF Pilot

```
</BODY>
</HTML>
```

Here we've returned to our pleasant pay raise calculator scenario. This time, we require that the user enter an employee name and a new pay rate (the current pay rate is filled in automatically with a hard-coded value). Two validation controls are on this page: a RequiredFieldValidator attached to the Employee Name field, and a RangeValidator connected to the New Rate field. It would be obnoxious to give users an error message if they failed to enter the employee's name, only to zap them with another error message when they neglect to enter a new pay rate; the validation summary gives them all the applicable error messages at once. To test this, navigate to the page and click the Save button without changing any of the values on the page.

The combination of red asterisks, warning messages on the page, and the message box alert probably makes this the most annoying data entry form in the history of Web programming, but it's really all in the name of science. Certainly your production Web forms will be more tasteful.

Using Multiple Validators

It often makes sense to have multiple validators attached to the same control. For example, if a field is required to be present and to fall within a certain range of values, you might attach both a RequiredFieldValidator and a RangeValidator to the control. There's no trick to this; simply assign as many validators as you need to implement the validation logic you want.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Handling Errors

Errors in ADO.NET are handled the same way as elsewhere in the .NET framework—you can catch exceptions that are thrown by objects involved in connecting to and interacting with the database.

The tricky thing about dealing with errors in data-access programming is that so many things can go wrong at various stages in each request. The programmer can phrase the command in a syntactically incorrect way; the network connection can be bad; the database server can be on vacation. So it's useful in database programming to have a rich collection of error messages from every element in every database call.

ADO.old provided this in the form of an Errors collection associated with the database connection object. In ADO.NET, you are given a provider-specific exception object that contains a collection of error messages. Again, the functionality is similar to ADO.old; the only difference is in the implementation details.

In the SQL Server provider, a SqlException object is thrown whenever a data-access operation fails. This object contains a collection of SqlError objects that you can examine in code to determine the full extent of what went wrong with your database call.

If you're interested only in determining whether a particular data-access operation succeeded or failed, you need only catch the top-level error. To get a brief error message pertaining to the error, use the Message property of the SqlException object. On the other hand, if you want complete and detailed information about what went wrong, you must iterate through the SqlError objects contained in the SqlException object (these are exposed through the SqlException object's Errors collection).

Listing 11.22 shows an example of the simple method of displaying a SQL error.

Listing 11.22 Displaying a Data-Access Error

```
<% @Page language='c#' debug='true' %>
<% @Import namespace='System.Data' %>
<% @Import namespace='System.Data.SqlClient' %>

<HTML>
<SCRIPT runat='server'>

  void Page_Load(Object Sender,EventArgs e)
  {
    SqlConnection cn;
    SqlDataAdapter da;
    DataSet ds;
    String strSQL;

    strSQL = "SELECT TOP 10 au_fname  FROM authors"  ;
    cn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs;");
    da = new SqlDataAdapter(strSQL, cn);

    // ** Fill DataSet
```

This document is created with the unregistered version of CHM2PDF Pilot

```
    ds = new DataSet();
    da.Fill(ds, "Authors");

      // ** Display data
      DataGrid1.DataSource = ds.Tables["Authors"].DefaultView;
      DataGrid1.DataBind();
  }

</SCRIPT>

<BODY>
  <ASP:datagrid id='DataGrid1' runat='server' />

</BODY>
</HTML>
```

This code intentionally contains an error (the SQL command contains a reference to a field that doesn't exist). To show how the code runs without the error condition, fix the SQL command (the string "SELECT * from authors" will work).

Listing 11.23 provides an example of the more complete way of iterating through the Errors collection to display complete error information.

Listing 11.23 Using the Errors Collection to Display Rich Error Information

```
try
{

  // ** Fill DataSet
  ds = new DataSet();
  da.Fill(ds, "Authors");

  // ** Display data
  DataGrid1.DataSource = ds.Tables["Authors"].DefaultView;
  DataGrid1.DataBind();
 }
catch(SqlException sx)
{
  foreach(SqlError se in sx.Errors)
  {
    Response.Write("SQL Error: " + se.Message + "<BR>");
  }
}
```

See the reference at the end of this chapter for more information on the properties and methods supported by the SqlError object. Also remember that SqlError is unique to the SQL Server–managed provider in ADO.NET (if you're using the OLE DB–managed provider, the analogous class is System.Data.OleDb.OleDbError).

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

ADO.NET Framework Reference

This section contains a brief reference to the ADO.NET objects mentioned in this chapter. It is not designed to be an exhaustive reference. For complete descriptions, including information on the objects not discussed in this chapter, refer to the .NET framework SDK documentation.

DataAdapter Class

Member of System.Data.Common. Abstract class. Inherits from System.ComponentModel.Component.

The DataAdapter class serves as the base class for data adapter implementations in the SQL Server, OLE DB (and potentially other) data providers in ADO.NET.

Note that this is not the class you instantiate when you want to access data in ADO.NET; to write data-access code using a data adapter in your applications, use the OleDbDataAdapter, SqlDataAdapter, or other provider-specific data adapter class.

Properties		
AcceptChangesDuringFill	DesignMode	MissingSchemaAction
Container	Events	Site
ContinueUpdateOnError	MissingMappingAction	TableMappings
Methods		
CloneInternals	FillSchema	GetType
CreateObjRef	Finalize	InitializeLifetimeService
CreateTableMappings	GetFillParameters	MemberwiseClone
Dispose	GetHashCode	ShouldSerializeTableMappings
Equals	GetLifetimeService	ToString
Fill	GetService	Update
Events		
Disposed		

DataSet Class

Member of System.Data. Inherits from System.ComponentModel.MarshalByValueComponent.

Note that the DataSet class is not owned by any particular provider; any ADO.NET data provider is capable of creating a DataSet object, and DataSet objects should interoperate across providers.

This document is created with the unregistered version of CHM2PDF Pilot

Properties		
CaseSensitive	EnforceConstraints	Namespace
Container	Events	Prefix
DataSetName	ExtendedProperties	Relations
DefaultViewManager	HasErrors	Site
DesignMode	Locale	Tables
Methods		
AcceptChanges	GetXml	ReadXmlSchema
Clear	GetXmlSchema	RejectChanges
Clone	HasChanges	Reset
Copy	InferXmlSchema	ResetTables
Dispose	MemberwiseClone	ShouldSerializeRelations
Equals	Merge	ShouldSerializeTables
Finalize	OnPropertyChanging	ToString
GetChanges	OnRemoveRelation	WriteXml
GetHashCode	OnRemoveTable	WriteXmlSchema
GetService	RaisePropertyChanging	
GetType	ReadXml	
Events		
Disposed	MergeFailed	

DataColumn Class

Member of System.Data. Inherits System.ComponentModel.MarshalByValueComponent.

The DataColumn object represents the structure, or schema, of data in a DataTable object. You use the DataColumn object to determine information about the structure of the field (not the data it contains).

This object is contained by the DataSet object and is therefore not provider specific.

Properties		
AllowDBNull	DataType	Ordinal
AutoIncrememt	DefaultValue	Prefix
AutoIncrementSeed	DesignMode	ReadOnly
AutoIncrementStep	Events	Site
Caption	Expression	Table
ColumnMapping	ExtendedProperties	Unique
ColumnName	MaxLength	
Container	Namespace	
Methods		
Dispose	GetHashCode	OnPropertyChanging

This document is created with the unregistered version of CHM2PDF Pilot

Dispose	GetService	RaisePropertyChanging
Equals	GetType	ToString
Finalize	MemberwiseClone	
Events		
Disposed		

DataRelation Class

Member of System.Data.

The DataRelation object is used to denote the relationship between two DataTable objects.

Properties		
AllowDBNull	DataType	Ordinal
ChildColumns	ExtendedProperties	ParentTable
ChildKeyConstraint	Nested	RelationName
ChildTable	ParentColumns	
DataSet	ParentKeyConstraint	
Methods		
CheckStateForProperty	GetHashCode	ToString
Equals	GetType	
Finalize	MemberwiseClone	

DataTable Class

Member of System.Data.

The DataTable object represents a unit of data arranged as collections of rows and columns. It is contained by the DataSet object and is not provider specific.

Properties		
CaseSensitive	DesignMode	Namespace
ChildRelations	DisplayExpression	ParentRelations
Columns	Events	Prefix
Constraints	ExtendedProperties	PrimaryKey
Container	HasErrors	Rows
DataSet	Locale	Site
DefaultView	MinimumCapacity	TableName
Methods		
AcceptChanges	Finalize	OnColumnChanged

This document is created with the unregistered version of CHM2PDF Pilot

BeginInit	GetChanges	OnColumnChanging
BeginLoadData	GetErrors	OnPropertyChanging
Clear	GetHashCode	OnRemoveColumn
Clone		OnRowChanged
Compute	GetService	OnRowChanging
Copy	GetType	OnRowDeleted
Dispose	ImportRow	OnRowDeleting
Dispose	LoadDataRow	RejectChanges
EndInit	MemberwiseClone	Reset
EndLoadData	NewRow	Select
Equals	NewRowFromBuilder	ToString
Events		
ColumnChanged	RowChanged	RowDeleting
ColumnChanging	RowChanging	
Disposed	RowDeleted	

OleDbCommand Class

Member of System.Data.OleDb. Inherits from System.Component.Component.

The OleDbCommand object is used to execute commands (including queries and such operations as data manipulation commands) against a data source.

Properties		
CommandText	Container	Parameters
CommandTimeout	DesignMode	Site
CommandType	DesignTimeVisible	Transaction
Connection	Events	UpdatedRowSource
Methods		
Cancel	ExecuteScalar	InitializeLifetimeService
CreateobjRef	Finalize	MemberwiseClone
CreateParameter	GetHashCode	Prepare
Dispose	GetLifetimeService	ResetCommandTimeout
Equals	GetService	ToString
ExecuteNonQuery	GetType	
ExecuteReader		
Events		
Disposed		

SqlDataAdapter Class

This document is created with the unregistered version of CHM2PDF Pilot

Member of System.Data.SqlClient. Inherits from System.Data.DataAdapter.

The SqlDataAdapter object is used primarily to create DataSet objects in the SQL Server–managed provider. In thick client context, the data adapter can also be used to provide cursorless navigation, filtering, creation, deletion, and updating of data. The OLE DB equivalent is System.Data.OleDb.OleDbDataAdapter.

Properties		
AcceptChangesDuringFill	Events	Site
Container	InsertCommand	TableMappings
ContinueUpdateOnError	MissingMappingAction	UpdateCommand
DeleteCommand	MissingSchemaAction	
DesignMode	SelectCommand	
Methods		
CloneInternals		InitializeLifetimeService
CreateObjRef		MemberwiseClone
CreateRowUpdatedEvent	FillSchema	OnFillError
CreateRowUpdatingEvent	Finalize	OnRowUpdated
CreateTableMappings	GetFillParameters	OnRowUpdating
Dispose	GetHashCode	ShouldSerializeTableMappings
	GetLifetimeService	ToString
Equals	GetService	
Fill	GetType	Update
Events		
AllowDBNull	DataType	Ordinal
Disposed	RowUpdated	
FillError	RowUpdating	

SqlDataReader Class

Member of System.Data.SqlClient. Inherits System.MarshalByRefObject.

The SqlDataReader object enables connection-based, fast access to data in the SQL Server–managed provider. The OLE DB equivalent to this object is System.Data.OleDb.OleDbDataReader.

Properties		
Depth	IsClosed	RecordsAffected
FieldCount	Item	
Methods		
Close	GetInt16	GetSqlInt64
CreateObjRef	GetInt32	GetSqlMoney
Equals	GetInt64	GetSqlSingle

This document is created with the unregistered version of CHM2PDF Pilot

Finalize	GetLifetimeService	GetSqlString
GetBoolean	GetName	GetSqlValue
GetByte	GetOrdinal	GetSqlValues
GetBytes	GetSchemaTable	GetString
GetChar	GetSqlBinary	GetType
GetChars		GetValue
GetDataTypeName	GetSqlBoolean	GetValues
GetDateTime	GetSqlByte	IntializeLifetimeService
GetDecimal	GetSqlDateTime	IsDBNull
GetDouble	GetSqlDecimal	MemberwiseClone
GetFieldType	GetSqlDouble	NextResult
GetFloat	GetSqlGuid	Read
GetGuid	GetSqlInt16	ToString
GetHashCode	GetSqlInt32	

SqlConnection Class

Member of System.Data.SqlClient.

The SqlConnection object is used to create and maintain a connection to a data source in the SQL Server–managed provider. Its OLE DB provider equivalent is System.Data.OleDb.OleDbConnection.

Properties		
ConnectionString	DataSource	ServerVersion
ConnectionTimeout	DesignMode	Site
Container	Events	State
Database	PacketSize	WorkstationId
Methods		
BeginTransaction	Equals	InitializeLifetimeService
ChangeDatabase	Finalize	MemberwiseClone
Close	GetHashCode	Open
CreateCommand	GetLifetimeService	ToString
CreateObjRef	GetService	
Dispose	GetType	
Events		
Disposed	InfoMessage	StateChange

SqlError Class

Member of System.Data.SqlClient.

This document is created with the unregistered version of CHM2PDF Pilot

The SqlError object is found in the Errors collection contained by the SqlException object. It is designed to provide SQL Server–specific error information. Its OLE DB provider equivalent is the OleDbError class.

Note that this object (and the SqlException object that contains it) is available only when you use the SQL Server–managed provider.

Properties		
Class	Number	Source
LineNumber	Procedure	State
Message	Server	
Methods		
Equals	GetHashCode	MemberwiseClone
Finalize	GetType	ToString

SqlException Class

Member of System.Data.SqlClient. Inherits from System.Exception.

The SqlException object is the object thrown when an error occurs in the ADO.NET SQL Server–managed provider. This object contains a collection of SqlError objects that you can use to get detailed information on exactly what caused the data-access error.

The equivalent to this class in the OLE DB provider is the OleDbException class.

This object is typically caught in an exception handler rather than constructed.

Properties		
Class	LineNumber	Source
Errors	Message	StackTrace
HelpLink	Number	State
Hresult	Procedure	TargetSite
InnerException	Server	
Methods		
Equals	GetHashCode	MemberwiseClone
Finalize	GetObjectData	ToString
GetBaseException	GetType	

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Copyright

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley were aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for special sales.

For more information, please contact:

Pearson Education Corporate Sales Division

201 W. 103rd Street

Indianapolis, IN 46290

(800) 428-5331

corpsales@pearsoned.com

Visit AW on the Web: www.awl.com/cseng/

Copyright 2002 by Pearson Education

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

05 04 03 02 DOC 4 3 2 1

First printing April 2002

This document is created with the unregistered version of CHM2PDF Pilot

Credits

Associate Publisher

Linda Engelman

Acquisitions Editor

Sondra Scott

Development Editors

Angela Allen

Laurie McGuire

Managing Editor

Charlotte Clapp

Project Editor

Carol Bowers

Copy Editor

Barbara Hacha

Indexer

Tim Tate

Proofreader

This document is created with the unregistered version of CHM2PDF Pilot

Jessica McCarty

Contributor

Anjani Chittajallu

Technical Editor

Joel Mueller

Team Coordinator

Lynne Williams

Media Developer

Dan Scherf

Interior Designer

Gary Adair

Cover Designer

Gary Adair

Page Layout

Ayanna Lacey

Dedication

For Celeste

This document is created with the unregistered version of CHM2PDF Pilot

Jeffrey P. McManus

This book is dedicated to my dad, who supported and encouraged me in everything I did.

Chris Kinsman

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Web directories
activating tracing for entire Web directories with Web.config 2nd
tracing 2nd
Web.config
activating tracing entire Web directories 2nd
!= (inequality operator), XPath queries
& (ampersand) character
XML
* (asterisk)
authorization settings (Web.Config/Machine.Config files)
* (asterisk) wildcard
HttpHandlers
-Y switch (XCOPY)
. (dot) operator
XPath queries
.FirstChild() property (XML) 2nd
.Load() method (XML) 2nd 3rd 4th 5th 6th
listings
10.11, loading a local XML file
10.12, loading an XML file residing on a Web server
10.13, loading a local XML file 2nd
.NET framework 2nd
Common Language Runtime (CLR) 2nd 3rd
/ (slash)
path attribute, forms authentication 2nd
/ (slash) character
XML closing tags
/* shortcut (XPath)
? (question mark)
authorization settings (Web.Config/Machine.Config files)
URL authorization
@ symbol
XPath queries (XML)
@Application directive (Global.aspx file) 2nd
@Assembly directive (Global.aspx file) 2nd 3rd
@Assembly directive (Page object)
@Control directive (Page object)
@Implements directive (Page object) 2nd
@Import directive (Page object) 2nd
@OutputCache directive (Page object)
@Page directive 2nd 3rd 4th 5th 6th
(Page object)
. [See also Page object, attributes]
listing 2.23, typical @Page directive 2nd
@Register directive (Page object)
~ (tilde) character
server side control HREFs

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

A switch (XCOPY)
AbortTransaction event
  Page object
AbsoluteExpiration argument (Cache class) 2nd 3rd
Access Control Lists (ACLs)
  file authorization 2nd
Accessibility attribute (PassportIdentity class)
accessing data 2nd
ACLs (Access Control Lists)
  file authorization 2nd
Activator .asmx file listing 2nd
Active Server Pages (ASP) 2nd 3rd 4th
  disadvantages 2nd 3rd 4th 5th 6th
Active-User array, displaying results in Global.asax files 2nd 3rd
ActiveX Data Objects. [See ADO]2nd [See ADO]
Add argument (Cache class)
AddDisplay.aspx code listing 2nd 3rd
ADO
  (ActiveX Data Objects) 2nd
ADO.NET 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
  classes
    DataAdapter 2nd 3rd
    DataColumn 2nd 3rd 4th 5th
    DataRelation 2nd
    DataSet 2nd
    DataTable 2nd
    OleDbCommand 2nd
    SqlConnection 2nd
    SqlDataAdapter 2nd
    SqlDataReader 2nd 3rd
    SqlError 2nd
    SqlException 2nd
  common database access scenarios 2nd 3rd
  connecting to databases 2nd 3rd 4th 5th 6th 7th 8th 9th
    error handling 2nd 3rd 4th
    OLE DB provider 2nd 3rd 4th
    Open/Close methods 2nd 3rd
    SQL Server provider 2nd
  creating Web Forms for data entry 2nd 3rd 4th
    comparison validation 2nd 3rd 4th 5th
    custom validation 2nd 3rd 4th 5th
    range validation 2nd 3rd 4th
    regular expression validation 2nd 3rd
    required field validation 2nd 3rd 4th
    summarizing validation errors 2nd 3rd 4th
    using multiple validators
    validation 2nd 3rd 4th
  data access 2nd
  data adapters 2nd 3rd 4th

This document is created with the unregistered version of CHM2PDF Pilot

. [See also data adapters]
binding DataView object to Web Forms controls 2nd 3rd 4th
binding objects to Web Forms controls 2nd 3rd 4th
displaying query data in browsers 2nd
expressing DataSets as XML 2nd 3rd 4th 5th
data-containing objects 2nd
disconnected access support 2nd 3rd
error handling 2nd 3rd 4th 5th 6th
new features 2nd 3rd 4th 5th 6th 7th
queries 2nd 3rd 4th
(begin)
(end)
command object 2nd
displaying query data in Web browsers 2nd 3rd 4th 5th
displaying query data in Web Forms DataGrid control 2nd
displaying query results using SqlDataAdapter and DataSet 2nd 3rd
executing stored procedures that return scalar values 2nd 3rd
passing parameters to stored procedures 2nd 3rd 4th 5th 6th
retrieving data reader objects via text commands 2nd 3rd 4th 5th 6th 7th
retrieving data via stored procedures 2nd 3rd 4th 5th
SqlDataReader object 2nd 3rd 4th 5th 6th 7th 8th
remoting support 2nd
server-side cursors 2nd
stored procedures
executing stored procedures that return data 2nd 3rd 4th 5th
passing parameters 2nd 3rd 4th 5th 6th
returning scalar values 2nd 3rd
versus text-based commands
XML support 2nd
AdRotator class 2nd 3rd 4th 5th 6th
AdvertisementFile property 2nd
banner dimensions
listing 2.29 2nd 3rd
listing 2.30 2nd
ads
ad insertion engine that replaces <adinsert tags with ads 2nd 3rd 4th
output from AdInserter.vb 2nd 3rd
advertisements
rotating (AdRotator class) 2nd 3rd 4th 5th 6th
banner dimensions
listing 2.29 2nd 3rd
listing 2.30 2nd
allowOverride attribute (Web.Config) 2nd
ampersand character (&)
XML
AND keyword
XPath queries
anonymous users, denying 2nd
AppendChild method (XML) 2nd
application directives (Global.asax file) 2nd 3rd 4th
application programming interface (API)
databases

This document is created with the unregistered version of CHM2PDF Pilot

databases. [See also ADO.NET]
Application-Scoped objects, declaring in Global.asax files 2nd 3rd
Application_AuthenticateRequest (global.asax) 2nd 3rd 4th
applications
databases. [See databases]
debugging 2nd 3rd
. [See also debugging]
initializing using Global.asax files 2nd
(begin)
(end)
application directives 2nd 3rd 4th
code declaration blocks 2nd 3rd
server-side includes 2nd
server-side object tags 2nd 3rd 4th 5th
tracing 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th
categorizing output 2nd 3rd 4th 5th 6th 7th
enabling application tracing 2nd 3rd 4th
enabling page tracing 2nd 3rd 4th
equipping pages for Trace mode 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
trace information categories 2nd
viewing application trace data from remote browser windows 2nd 3rd
Web.Config/Machine.Config Trace settings 2nd 3rd 4th
AppSettings method (ConfigurationSettings object)
ArrayList
Web services returning array lists 2nd 3rd 4th 5th
ArrayList object
binding DataGrid control 2nd
listing 2.34, binding DropDownList control to data in ArrayList object 2nd 3rd
arrays
Active-User array, displaying results in Global.asax files 2nd 3rd
ASMX file extension
ASP (Active Server Pages) 2nd 3rd 4th
disadvantages 2nd 3rd 4th 5th 6th
ASP.NET 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
advantages 2nd 3rd 4th 5th
control model 2nd 3rd
(begin)
(end)
event handling 2nd 3rd 4th 5th 6th 7th
event handling. [See also event handling]
form state 2nd 3rd
HTML controls 2nd
HTML controls. [See also HTML controls]
server controls 2nd 3rd 4th 5th
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
events 2nd 3rd 4th 5th
. [See also events, event handling]
new features 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
State Server 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
Web services. [See Web services]
AspCompat attribute, Page object 2nd
assemblies

This document is created with the unregistered version of CHM2PDF Pilot

@Assembly directive (Global.aspx file) 2nd 3rd
asterisk (*)
authorization settings (Web.Config/Machine.Config files)
asterisk (*) wildcard
HttpHandlers
asynchronous clients, Web services 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
attributes
form authentication settings 2nd 3rd 4th 5th
Page object 2nd 3rd
AspCompat 2nd
AutoEventWireup 2nd
Buffer 2nd
ClassName
ClientTarget 2nd
CodePage 2nd
CompilerOptions 2nd
ContentType 2nd
Culture 2nd
Debug
Description
EnableSessionState
EnableViewState 2nd
EnableViewStateMac 2nd
ErrorPage
Inherits
Language
LCID 2nd
SmartNavigation
Src 2nd
Trace
TraceMode
Transaction
UICulture
WarningLevel 2nd
Passport authentication 2nd
WebMethod( ) 2nd 3rd 4th
calling via HTTP GET
Simple() WebMethod of SimpleDataSet 2nd 3rd 4th 5th
viewing WSDL output
WebMethodAttribute class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
WebMethodAttribute class;BufferResponse property 2nd
WebMethodAttribute class;CacheDuration property 2nd 3rd 4th
WebMethodAttribute class;Description property 2nd 3rd 4th 5th 6th 7th
WebMethodAttribute class;EnableSession property 2nd 3rd 4th 5th
WebMethodAttribute class;TransactionOption property 2nd
XML 2nd 3rd
creating 2nd
displaying attribute values using GetAttribute method 2nd 3rd
using attributes in queries 2nd 3rd 4th
authentication
cookie-based authentication
file authorization 2nd

This document is created with the unregistered version of CHM2PDF Pilot

forms authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
advantages 2nd
process flow 2nd 3rd 4th 5th
roles 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
settings 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
versus Windows authentication 2nd 3rd
HttpModules 2nd 3rd 4th
Passport authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th
Passport Web site
PassportIdentity class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
Site IDs
web.config setup 2nd
URL authorization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
location element 2nd 3rd 4th 5th
using forms authentication with URL authorization 2nd 3rd 4th
FormsAuthenticationTicket class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
Web.Config/Machine.Config settings 2nd 3rd 4th
Windows authentication 2nd 3rd
versus forms authentication 2nd 3rd
WindowsBuiltInRole enumeration 2nd 3rd 4th 5th 6th 7th
authorization
groups
Web.Config/Machine.Config settings 2nd
AuthorizeRequest event
AutoEventWireup attribute, Page object 2nd
availability 2nd

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

banners
rotating advertisements (AdRotator class) 2nd 3rd 4th 5th 6th 7th 8th 9th
banner dimensions
listing 2.29 2nd 3rd
listing 2.30 2nd 3rd
BDay_precision attribute (PassportIdentity class)
BeginRequest HttpModule 2nd 3rd 4th 5th
(time stamping output HTML)
binding data
Repeater control 2nd 3rd 4th 5th 6th 7th
Web forms
binding data. [See data binding]
binding date
DataBinding event
Page object
binding objects to Web Forms controls via data adapters 2nd 3rd 4th
DataView object 2nd 3rd 4th
binding server controls to data 2nd
binding Web services to forms 2nd 3rd 4th 5th
Birthdate attribute (PassportIdentity class)
book.xls file
listing 10.43, books.xml File Containing a Link to the books.xsl Style Sheet 2nd
book.xml document
listings
10.33, simplified book.xml document 2nd
books.xml document
inserting new books using the XmlNode InsertAfter method 2nd 3rd
books.xml file
creating file with XmlTextWriter object 2nd 3rd 4th 5th
querying
combining multiple criteria with AND and OR 2nd
excluding nodes based on attribute values 2nd
retrieving a specific BOOK node by querying on text in BookÕs TITLE node 2nd
retrieving a specific TITLE node by querying on its text 2nd
retrieving author information based on author ID 2nd
retrieving multiple instances of same author 2nd 3rd
running a simple query 2nd
XPath query page used to test XPath queries against books.xml 2nd
books.xml listing 2nd 3rd
BreadCrumbSearch.aspx code listing 2nd 3rd 4th
breakpoints 2nd
browers
ContentType attribute (Page object) 2nd
browsers
capabilities settings
Web.Config/Machine.Config settings 2nd
determining browser capabilities 2nd 3rd 4th 5th 6th 7th
HttpBrowserCapabilities class
caching 2nd 3rd 4th

This document is created with the unregistered version of CHM2PDF Pilot

page requests 2nd 3rd 4th 5th 6th
specifying browser type via ClientTarget attribute 2nd
Buffer attribute, Page object 2nd
buffering 2nd
BufferResponse property (WebMethodAttribute class) 2nd
business logic componenets
Global Assembly Cache (GAC) 2nd 3rd 4th
Button class 2nd 3rd 4th
Button.aspx listing 2nd 3rd 4th 5th 6th
buttons
HtmlButton control 2nd 3rd 4th 5th
HtmlInputButton control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputRadioButton control 2nd
ImageButton control 2nd 3rd 4th
LinkButton control 2nd 3rd
RadioButton control 2nd 3rd 4th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

C switch (XCOPY)
Cache class 2nd 3rd 4th 5th
. [See also caching]
AbsoluteExpiration argument 2nd 3rd
Add argument
caching database queries 2nd 3rd 4th 5th 6th 7th
Count argument 2nd
Insert argument
ItemPriority argument
MaxItems argument 2nd
SlidingExpiration argument 2nd 3rd 4th 5th
Cache5.aspx code listing 2nd
CacheCallback.aspx code listing 2nd 3rd 4th
CachedBrowser.aspx code listing
CacheDependency class 2nd 3rd 4th 5th 6th 7th 8th 9th
CacheDependency.aspx code listing 2nd 3rd
CachedHost.aspx code listing 2nd
CachedSearchBroken.aspx code listing 2nd 3rd 4th
CachedSearchFixed.aspx code listing 2nd 3rd 4th
CacheDuration property
WebMethodAttribute class 2nd 3rd 4th
CacheExpiration.aspx code listing 2nd 3rd 4th 5th 6th
caches
Global Assembly Cache (GAC) 2nd 3rd 4th 5th
gacutil utility 2nd 3rd
CacheTerritories code listing 2nd 3rd
caching 2nd 3rd 4th
@OutputCache directive (Page object)
browser differences 2nd 3rd 4th
database queries 2nd 3rd 4th 5th 6th 7th
dependencies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
expiring items 2nd 3rd 4th 5th
form results 2nd 3rd 4th 5th 6th
HttpCachePolicy class 2nd
HttpCacheVaryByHeaders class 2nd
HttpCacheVaryByParams class 2nd
listings
5.11, directive for 5-minute cache 2nd
5.12, browser capabilities 2nd
5.13, form using RequiredFieldValidator 2nd
5.14, form creating copies for different browsers 2nd 3rd
5.15, creating cached page for each header 2nd
5.16, incorrect form output caching 2nd 3rd 4th
5.17, correct form output caching 2nd 3rd 4th
5.18, partial page caching user control (SalesPeople.ascx) 2nd 3rd
5.19, partial page caching user control (SalesByEmployee.ascx) 2nd 3rd 4th 5th
5.20, partial page caching of previous search terms (LastFiveSearchTerms) 2nd 3rd
5.21, search page containing term caching (BreadCrumbSearch.aspx) 2nd 3rd 4th
5.22, containing page defining cache directive 2nd

This document is created with the unregistered version of CHM2PDF Pilot

5.22, containing page without cache directive
5.23, containing page without cache directive
5.24, query that places output into drop-downlist 2nd 3rd
5.25, query that stores output in cache 2nd 3rd 4th
5.26, expiring items 2nd 3rd
5.27, re-adding dropped out cache items 2nd 3rd 4th
5.28, refreshing cached items via dependencies 2nd 3rd
5.29, removing/inserting items from cache with expirations 2nd 3rd 4th 5th
output caching (entire pages) 2nd 3rd 4th 5th 6th 7th
partial pages 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
allowed user control/ output caching combinations 2nd 3rd 4th 5th
generating crumb trails of previous search terms 2nd 3rd 4th 5th
generating phone lists 2nd 3rd 4th 5th 6th
re-adding dropped out items via callback routine 2nd 3rd 4th 5th
removing expired items 2nd
VaryByCustom/VaryByHeader 2nd 3rd
caching output (HttpModules) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
Calendar class 2nd 3rd 4th 5th
Calendar Web control 2nd 3rd 4th 5th 6th 7th 8th
output
rendered HTML source 2nd 3rd 4th 5th 6th 7th
callback routines
re-adding dropped out cache items 2nd 3rd 4th 5th
categorizing Trace output 2nd 3rd 4th 5th 6th 7th
cells
HtmlTableCell control 2nd
HtmlTableCellCollection control 2nd
HtmlTextArea control 2nd
cells. [See tables]
characters
XML
CDATA elements 2nd 3rd
character entities 2nd 3rd 4th
enclosing character data 2nd 3rd 4th 5th
check boxes
HtmlInputCheckbox control 2nd
CheckBox class 2nd 3rd 4th
ChildNodes class (XML) 2nd 3rd 4th 5th
City attribute (PassportIdentity class)
classes
. [See also objects]
AdRotator 2nd 3rd 4th 5th 6th
AdvertisementFile property 2nd
banner dimensions
listing 2.29 2nd 3rd
listing 2.30 2nd
Button 2nd 3rd 4th
Cache 2nd 3rd 4th 5th
. [See also caching]
AbsoluteExpiration argument 2nd 3rd
Add argument
caching database queries 2nd 3rd 4th 5th 6th 7th

This document is created with the unregistered version of CHM2PDF Pilot

Count argument 2nd
Insert argument
ItemPriority argument
MaxItems argument 2nd
SlidingExpirationargument 2nd 3rd 4th 5th
CacheDependency 2nd 3rd 4th 5th 6th 7th 8th 9th
Calendar 2nd 3rd 4th 5th
CheckBox 2nd 3rd 4th
ChildNodes (XML) 2nd 3rd 4th 5th
compiling into .NET components 2nd
ConfigurationSettings
AppSettings method
Control 2nd 3rd
DataAdapter (ADO.NET) 2nd 3rd
DataColumn (ADO.NET) 2nd 3rd 4th 5th
DataGrid 2nd 3rd
DataList 2nd 3rd 4th 5th 6th
DataRelation (ADO.NET) 2nd
DataSet (ADO.NET) 2nd
DataTable (ADO.NET) 2nd
Debug 2nd 3rd
DropDownList 2nd 3rd 4th 5th 6th 7th 8th
EventLog 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
FormsAuthenticationTicket 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
GenericIdentity 2nd 3rd
GenericPrincipal 2nd
HtmlAnchor 2nd 3rd 4th 5th
HtmlButton 2nd 3rd 4th 5th
HtmlForm 2nd 3rd 4th 5th 6th 7th 8th 9th
HtmlImage 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
HtmlInputButton 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputCheckbox 2nd
HtmlInputFile 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputHidden 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputImage 2nd
HtmlInputRadioButton 2nd
HtmlInputText 2nd
HtmlSelect 2nd
HtmlTable 2nd
HtmlTableCell 2nd
HtmlTableCellCollection 2nd
HtmlTableRow 2nd
HtmlTableRowCollection 2nd
HtmlTextArea 2nd
HttpApplication 2nd 3rd 4th 5th 6th 7th 8th
HttpBrowserCapabilities 2nd 3rd 4th 5th 6th 7th
caching 2nd 3rd 4th
HttpCachePolicy 2nd
HttpCacheVaryByHeaders 2nd
HttpCacheVaryByParams 2nd
HttpRequest 2nd
HttpResponse 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

HttpServerUtility 2nd
HttpSessionState 2nd 3rd 4th 5th
. [See also state management]
TimeOut property 2nd
Hyperlink 2nd 3rd
IHttpHandler 2nd
. [See also HttpHandlers]2nd [See also HttpHandlers]
IHttpModule 2nd
IhttpModule
. [See also HttpModules]
Image 2nd 3rd
ImageButton 2nd 3rd 4th
inheritance
Inherits attribute (Page object)
Label 2nd 3rd
LinkButton 2nd 3rd 4th 5th 6th 7th
ListBox 2nd 3rd
OleDbCommand 2nd 3rd
OleDbCommand (ADO.NET) 2nd
OleDbConnection 2nd 3rd 4th 5th 6th
OleDbDataReader
Page 2nd 3rd
Panel 2nd 3rd
PassportIdentity 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
attributes 2nd
PerformanceCounter 2nd
Repeater 2nd 3rd 4th 5th 6th 7th
SmtpMail 2nd 3rd
SoapHttpClientProtocol 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
CookieContainer property 2nd 3rd 4th 5th 6th 7th 8th 9th
SqlCommand
SQLCommand
ExecuteScalar method 2nd 3rd
SqlCommand
Parameters collection 2nd
SqlConnection 2nd 3rd 4th 5th 6th 7th
SqlConnection (ADO.NET) 2nd
SqlDataAdapter
. [See also data adapters (ADO.NET)]
SqlDataAdapter (ADO.NET) 2nd
SqlDataReader 2nd 3rd 4th 5th 6th 7th 8th
calling stored procedures 2nd 3rd 4th
returning SqlDataReader objects via text commands 2nd 3rd 4th 5th 6th 7th
SqlDataReader(ADO.NET) 2nd 3rd
SqlError (ADO.NET) 2nd
SqlException (ADO.NET) 2nd
Table 2nd
TableCell 2nd 3rd
TableRow 2nd 3rd
TextBox 2nd
TraceContext 2nd
ValidationEventArgs 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

WebControl 2nd 3rd
WebMethodAttribute class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
BufferResponse property 2nd
CacheDuration property 2nd 3rd 4th
Description property 2nd 3rd 4th 5th 6th 7th
EnableSession property 2nd 3rd 4th 5th
TransactionOption property 2nd
WebService 2nd 3rd
WindowsIdentity 2nd
WindowsPrincipal 2nd 3rd
XML Document Object Model (DOM) 2nd 3rd 4th 5th 6th 7th 8th 9th
extended classes 2nd
fundamental classes 2nd 3rd
XmlDataReader 2nd 3rd 4th 5th 6th 7th
XmlDocument 2nd 3rd 4th 5th 6th 7th 8th
XMLDocument class
creating XmlNodeReader objects from XmlDocument objects
XmlNamedNodeMap 2nd
XmlNode 2nd 3rd 4th
InsertAfter method 2nd 3rd
viewing document data 2nd 3rd 4th 5th 6th 7th 8th
XmlNodeList 2nd
XmlNodeReader 2nd 3rd 4th
navigating/updating documents 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
XmlReader 2nd
XmlTextReader 2nd 3rd
XmlTextWriter 2nd
writing data 2nd 3rd 4th 5th 6th 7th
XmlValidatingReader 2nd 3rd
validating schemas 2nd 3rd 4th 5th 6th 7th 8th
XmlWriter 2nd 3rd
ClassName attribute, Page object
clients
asynchronous clients, Web services 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
ClientTarget attribute, Page object 2nd
Close method
database connections 2nd 3rd
CLR (Common Language Runtime) 2nd 3rd
CLS (Common Language Specification) 2nd 3rd
code
breakpoints 2nd
debugging. [See debugging]
Global.asax declaration blocks 2nd 3rd
reusing 2nd 3rd
custom user-interface objects. [See controls]
server-side includes (SSIs) 2nd
reusing code. [See components]
code behind 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
denoting code-behind class used by a page (ClassName attribute)
inheriting from a class using Inherits attribute (Page object)
listing 2.24, definition of a code-behind Page_Load event procedure without AutoEventWireup 2nd
listing 2.8, SamplePage3.aspx, using code behind 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

listing 2.9, SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd
Src (source) attribute (Page object) 2nd
src attribute (Page object)
Code Behind Web Service Class That Implements a Cached Time Service 2nd
code listings. [See listings]
code, separating from presentation using code behind 2nd 3rd 4th 5th 6th 7th 8th 9th
. [See also code behind]
listing 2.8, SamplePage3.aspx, using code behind 2nd 3rd
listing 2.9, SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd
CodePage attribute, Page object 2nd
COM (Component Object Model) libraries 2nd
COM objects 2nd
. [See also assemblies]
comma-delimited data (XML) 2nd 3rd
enclosing character data 2nd 3rd 4th 5th
CDATA elements 2nd 3rd
character entities 2nd 3rd 4th
command line
creating .NET components with command-line compiler 2nd 3rd 4th 5th 6th
gacutil tool 2nd 3rd
commands
ADO.NET command objects 2nd
calling stored procedures using SqlCommand 2nd 3rd 4th 5th 6th 7th
executing commands that donÕt return data (nonquery commands) 2nd 3rd 4th
executing text-based commands to retrieve data reader objects 2nd 3rd 4th 5th 6th 7th
COPY
Windows 2000 changes
data-manipulation commands
MOVE
Windows 2000 changes
SqlCommand class
Parameters collection 2nd
SQLCommand object
ExecuteScalar method 2nd 3rd
XCOPY. [See XCOPY deployment]
CommitTransaction event
Page object
Common Language Runtime (CLR) 2nd 3rd
Common Language Specification (CLS) 2nd 3rd
CompareValidator control 2nd 3rd 4th 5th
properties
ValidationCompareOperator enumeration 2nd 3rd
compilation settings (Web.Config/Machine.Config files) 2nd 3rd
compiler warnings (WarningLevel attribute, Page object) 2nd
CompilerOptions attribute, Page object 2nd
compiling controls as .NET components 2nd 3rd 4th 5th 6th
creating components with command-line compiler 2nd 3rd 4th 5th 6th
deploying components 2nd
complex types (XML) 2nd 3rd 4th 5th 6th 7th
componenets
Global Assembly Cache (GAC) 2nd 3rd 4th
Component Object Model (COM) libraries 2nd

This document is created with the unregistered version of CHM2PDF Pilot

components
creating .NET components with Visual Studio
composite server controls 2nd 3rd 4th 5th 6th 7th 8th
Config.Web files
reading with AppSettings method (ConfigurationSettings object)
configuration 2nd 3rd 4th 5th 6th
. [See also Web.Config files]
accessing configuration files programmatically 2nd 3rd 4th
AppSettings method (ConfigurationSettings object)
ASP.old versus ASP.NET 2nd 3rd
editing Web configuration files in Visual Studio.NET 2nd
global/local configuration files 2nd
locking down Web configuration settings 2nd
Machine.Config file 2nd
multiple locations 2nd 3rd 4th 5th 6th
section handlers
authentication 2nd 3rd 4th
authorization 2nd
browser capabilities 2nd
compilation 2nd 3rd
custom errors 2nd 3rd
execution timeout 2nd
globalization 2nd 3rd
HTTP handlers
HTTP modules 2nd
pages 2nd 3rd
process model 2nd 3rd
session state 2nd 3rd
trace 2nd 3rd 4th
Web services 2nd 3rd
section handlers (Web.Config/Machine.Config files)
Web.Config files 2nd
. [See also Web.Config files]
ConfigurationSettings object
AppSettings method
confirmations, file deletion (XCOPY) 2nd 3rd
connecting to databases 2nd 3rd 4th 5th 6th 7th 8th 9th
error handling 2nd 3rd 4th
OLE DB provider 2nd 3rd 4th
Open/Close methods 2nd 3rd
SQL Server provider 2nd
constructors
CacheDependency class
consuming Web services 2nd 3rd 4th 5th 6th 7th 8th 9th
asynchronous clients 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
SoapHttpClientProtocol class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
CookieContainer property 2nd 3rd 4th 5th 6th 7th 8th 9th
ContainerPage.aspx code listing 2nd
ContentType attribute, Page object 2nd
control
registering
@Register directive (Page object)

This document is created with the unregistered version of CHM2PDF Pilot

Control class 2nd 3rd
Control directive
user controls 2nd
controls
@Control directive (Page object)
ASP.NET control model 2nd 3rd
(begin)
(end)
event handling 2nd 3rd 4th 5th 6th 7th
event handling. [See also event handling]
form state 2nd 3rd
HTML controls 2nd
HTML controls. [See also HTML controls]
server controls 2nd 3rd 4th 5th
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
composite controls 2nd 3rd 4th 5th 6th 7th
creating basic server controls
Register directive 2nd
delegation 2nd 3rd 4th
HTML controls 2nd 3rd
equivalent HTML tags 2nd 3rd
HtmlAnchor 2nd 3rd 4th 5th
HtmlButton 2nd 3rd 4th 5th
HtmlForm 2nd 3rd 4th 5th 6th 7th 8th 9th
HtmlImage 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
HtmlInputButton 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputCheckbox 2nd
HtmlInputFile 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputHidden 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputImage 2nd
HtmlInputRadioButton 2nd
HtmlInputText 2nd
HtmlSelect 2nd
HtmlTable 2nd
HtmlTableCell 2nd
HtmlTableCellCollection 2nd
HtmlTableRow 2nd
HtmlTableRowCollection 2nd
HtmlTextArea 2nd
server controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
binding controls to data 2nd
Calendar class 2nd 3rd 4th 5th
CheckBox class 2nd 3rd 4th
compiling controls as .NET components 2nd 3rd 4th 5th 6th
Control class 2nd 3rd
creating basic server controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
creating composite controls 2nd 3rd 4th 5th 6th 7th 8th
DataGrid class 2nd 3rd
DataList class 2nd 3rd 4th 5th 6th
DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
events 2nd 3rd 4th
generating postback 2nd 3rd 4th 5th

This document is created with the unregistered version of CHM2PDF Pilot

HTML controls 2nd
HTML controls. [See also HTML controls]
Hyperlink class 2nd 3rd
Image class 2nd 3rd
ImageButton class 2nd 3rd 4th
LinkButton class 2nd 3rd
ListBox class 2nd 3rd
Panel class 2nd 3rd
persistence support 2nd 3rd 4th 5th 6th 7th
RadioButton class 2nd 3rd 4th
Repeater class 2nd 3rd 4th 5th 6th 7th
rich clients 2nd 3rd 4th
subclassing existing controls 2nd 3rd 4th 5th
supporting designers 2nd
Table class 2nd
TableCell class 2nd 3rd
TableRow class 2nd 3rd
TextBox class 2nd
validation controls 2nd
validation controls. [See also validation controls]
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
subclassing 2nd 3rd 4th 5th
user controls 2nd 3rd 4th 5th 6th 7th 8th 9th
adding methods 2nd 3rd 4th 5th
adding properties 2nd 3rd 4th
Control directive 2nd
handling events 2nd 3rd
programming controls in pages 2nd 3rd 4th
Register directive 2nd 3rd
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th 20th 21st 22nd 23rd 24th 25th 26th
AdRotator class 2nd 3rd 4th 5th 6th
Button class 2nd 3rd 4th
Calendar class 2nd 3rd 4th 5th
CheckBox class 2nd 3rd 4th
data binding
data binding. [See also data binding]
DataGrid class 2nd 3rd
DataList class 2nd 3rd 4th 5th 6th
determining browser capabilities 2nd 3rd 4th 5th 6th 7th
DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
equivalent HTML tags 2nd 3rd
event model 2nd 3rd
HttpApplication class 2nd 3rd
HttpBrowserCapabilities class 2nd 3rd 4th 5th 6th 7th
HttpRequest class 2nd
HttpResponse class 2nd 3rd
HttpServerUtility class 2nd
HttpSessionState class 2nd 3rd
Hyperlink class 2nd 3rd
Image class 2nd 3rd
ImageButton class 2nd 3rd 4th

This document is created with the unregistered version of CHM2PDF Pilot

Label class 2nd 3rd
LinkButton class 2nd 3rd
ListBox class 2nd 3rd
mobile controls 2nd
Panel class 2nd 3rd
postback 2nd 3rd 4th
programming (creating an ASP.NET Web form) 2nd
RadioButton class 2nd 3rd 4th
Repeater class 2nd 3rd 4th 5th 6th 7th
Table class 2nd
TableCell class 2nd 3rd
TableRow class 2nd 3rd
TextBox class 2nd
View State 2nd 3rd 4th
WebControl class 2nd 3rd
cookie-based authentication
cookieless session management
rewriting paths via HttpModule 2nd 3rd 4th 5th 6th
cookies
FormsAuthenticationTicket class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
RFC 2109
state management 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
cookie-based session identity 2nd 3rd
cookieless session identity 2nd 3rd 4th 5th 6th
Web services 2nd 3rd 4th 5th 6th 7th 8th 9th
COPY command
Windows 2000 changes
copying files
XCOPY. [See XCOPY deploymnet]
Count argument (Cache class) 2nd
counters
server controls
page that contains counter control 2nd 3rd
counters (HttpHandlers) 2nd 3rd 4th 5th 6th 7th 8th
Country attribute (PassportIdentity class)
culture
UICulture attribute, Page object
Culture attribute, Page object 2nd
cursors
ADO.NET support for server-side cursors 2nd
custom performance monitors 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
creating monitor categories 2nd 3rd 4th 5th 6th
deleting monitor categories 2nd 3rd
sending application information to monitors via PerformanceCounter object 2nd 3rd 4th 5th
Windows Performance Monitor utility 2nd
custom server controls 2nd 3rd 4th
binding controls to data 2nd
compiling controls as .NET components 2nd 3rd 4th 5th 6th
creating components with command-line compiler 2nd 3rd 4th 5th 6th
deploying components 2nd
creating basic server controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
creating composite controls 2nd 3rd 4th 5th 6th 7th 8th

This document is created with the unregistered version of CHM2PDF Pilot

events 2nd 3rd 4th
generating postback 2nd 3rd 4th 5th
persistence support 2nd 3rd 4th 5th 6th 7th
registering controls with @Register directive 2nd
rich clients 2nd 3rd 4th
subclassing existing controls 2nd 3rd 4th 5th
supporting designers 2nd
validation controls 2nd 3rd 4th 5th 6th
CompareValidator 2nd 3rd 4th 5th
CustomValidator 2nd 3rd 4th 5th
multiple validators
RangeValidator 2nd 3rd 4th
RegularExpressionValidator 2nd 3rd
RequiredFieldValidator 2nd 3rd 4th
summarizing validation errors 2nd 3rd 4th
customErrors section (Web.Config/Machine.Config files) 2nd 3rd
CustomTextBox control 2nd
ASP.NET Page that uses an instance of subclassed CustomText control 2nd 3rd
CustomValidator control 2nd 3rd 4th 5th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

D switch (XCOPY) 2nd 3rd
data
  XML
    abbreviated close-tag syntax 2nd 3rd
    accessing data. [See XML, data access] 2nd [See XML, data access]
    CDATA elements 2nd 3rd
    character entities 2nd 3rd 4th
    delimited format 2nd 3rd
    elements that contain data 2nd 3rd 4th 5th
    enclosing character data 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
    navigating/updating documents with XmlNodeReader object 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
    querying. [See XML, XPath queries]
    using XmlDataReader object 2nd 3rd 4th 5th 6th 7th
    viewing document data with XmlNode object 2nd 3rd 4th 5th 6th
    writing with XmlTextWriter object 2nd 3rd 4th 5th 6th 7th
data access 2nd
data adapters (ADO.NET) 2nd 3rd 4th 5th 6th
  binding objects to Web Forms controls 2nd 3rd 4th
    DataView object 2nd 3rd 4th
    expressing DataSets as XML 2nd 3rd
  creating DataSet object 2nd 3rd 4th
  expressing DataSets as XML 2nd 3rd 4th 5th
data binding 2nd 3rd 4th
  ASP.NET 2nd
    binding objects to Web Forms controls 2nd 3rd 4th 5th 6th 7th 8th
  Repeater control 2nd 3rd 4th 5th 6th 7th
  Web forms
data reader objects
  SqlDataReader 2nd
    calling stored procedures 2nd 3rd 4th
    returning via text commands 2nd 3rd 4th 5th 6th 7th
data representation 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
  advertisement 2nd
  HTML versus XML
    HTML 2nd 3rd 4th 5th 6th 7th 8th 9th
    XML 2nd 3rd
  location information 2nd 3rd
data-manipulation commands
DataAdapter class (ADO.NET) 2nd 3rd
databases
  ADO.NET 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
    common database access scenarios 2nd 3rd
    data access 2nd
    disconnected access support 2nd 3rd 4th
    new features in ADO.NET 2nd 3rd 4th 5th 6th 7th
    remoting support 2nd
    server-side cursors 2nd
    XML support 2nd
  connecting to databases 2nd 3rd 4th 5th 6th 7th 8th 9th

This document is created with the unregistered version of CHM2PDF Pilot

error handling 2nd 3rd 4th
OLE DB provider 2nd 3rd 4th
Open/Close methods 2nd 3rd
SQL Server provider 2nd
creating Web Forms for data entry 2nd 3rd 4th
comparison validation 2nd 3rd 4th 5th
custom validation 2nd 3rd 4th 5th
range validation 2nd 3rd 4th
regular expression validation 2nd 3rd
required field validation 2nd 3rd 4th
summarizing validation errors 2nd 3rd 4th
using multiple validators
validation 2nd 3rd 4th
data adapters 2nd 3rd 4th
binding DataView object to Web Forms controls 2nd 3rd 4th
binding objects to Web Forms controls 2nd 3rd 4th
creating DataSet object 2nd 3rd 4th
displaying query data in browsers 2nd
expressing DataSets as XML 2nd 3rd 4th 5th 6th 7th 8th
error handling 2nd 3rd 4th 5th 6th
listings
11.14, data-entry form 2nd 3rd 4th
11.15, required field validation 2nd 3rd
11.16, required field funcion, client-side validation 2nd
11.17, comparison validation 2nd 3rd
11.18, range validation 2nd 3rd
11.19, regular expression validation 2nd 3rd
11.20, custom validation 2nd 3rd
11.21, displaying validation error summary 2nd 3rd 4th
11.22, displaying data access errors 2nd 3rd
11.23, using Errors collection to display rich error information 2nd
12.1, opening a connection using SQL Server provider 2nd 3rd
12.10, displaying query results using SqlDataAdapter and DataSet 2nd 3rd
12.11, displaying query data in Web Forms DataGrid control 2nd 3rd
12.12, binding DataGrid control to ArrayList object 2nd
12.13, displaying query data in Web browsers 2nd 3rd
12.2, opening a connection using OLE DB provider 2nd
12.3, opening a connection with error handling 2nd 3rd
12.4, executing a text command that returns a SqlDataReader object 2nd 3rd
12.5, calling a simple stored procedure using SqlCommand 2nd 3rd
12.6, calling a parameterized stored procedure using SqlCommand Parameters collection 2nd 3rd
12.7, calling a parameterized stored procedure 2nd 3rd 4th
12.8, executing nonquery commands 2nd 3rd
12.9, ExecuteScalar method 2nd
queries 2nd 3rd 4th
(begin)
(end)
caching 2nd 3rd 4th 5th 6th 7th
command objects 2nd
displaying query data in Web browsers 2nd 3rd 4th 5th
displaying query data in Web Forms DataGrid control 2nd
displaying query results using SqlDataAdapter and DataSet 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

executing stored procedures that return scalar values 2nd 3rd
passing parameters to stored procedures 2nd 3rd 4th 5th 6th
retrieving data reader objects via text commands 2nd 3rd 4th 5th 6th 7th
retrieving data via stored procedures 2nd 3rd 4th 5th
SqlDataReader object 2nd 3rd 4th 5th 6th 7th 8th
stored procedures
executing stored procedures that return data 2nd 3rd 4th 5th
passing parameters 2nd 3rd 4th 5th 6th
returning scalar values 2nd 3rd
versus text-based commands
DataBinding event
Page object
Web controls 2nd
DataBinding event (custom server controls)
DataColumn class (ADO.NET) 2nd 3rd 4th 5th
DataColumn object
ADO.NET
DataGrid class 2nd 3rd
DataGrid control
displaying query data 2nd 3rd 4th
DataList class 2nd 3rd 4th 5th 6th
listing 2.33, DataList server control bound to a Hashtable object 2nd 3rd
DataReader object
ADO.NET
DataRelation class (ADO.NET) 2nd
DataRow object
ADO.NET
DataSet
Web services 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th
. [See also SimpleDataSet]
WebMethod returning a DataTable 2nd 3rd 4th 5th
WebMethod returning a Master-Detail relationship 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
DataSet class (ADO.NET) 2nd
DataSet object
ADO.NET 2nd
creating via data adapters 2nd 3rd 4th
expressing as XML 2nd 3rd 4th 5th 6th 7th 8th
DataTable class (ADO.NET) 2nd
DataView object
binding to Web Forms 2nd 3rd 4th
Debug class 2nd 3rd
Debug object 2nd 3rd
debugging 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
activating via Debug attribute (Page object)
breakpoints 2nd
custom performance monitors 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
creating monitor categories 2nd 3rd 4th 5th 6th
deleting monitor categories 2nd 3rd
sending application information to monitors via PerformanceCounter object 2nd 3rd 4th 5th
Windows Performance Monitor utility 2nd
debug page output 2nd
enabling

This document is created with the unregistered version of CHM2PDF Pilot

application level 2nd 3rd
Debug object 2nd 3rd 4th
page level 2nd 3rd 4th
ErrorPage attribute (Page object)
listings
3.4, test page for debugging 2nd 3rd
3.5, Web.config file that activates application-level debugging 2nd
3.6, calling Debug.Write to display debug information in development environment 2nd
tracing 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th
categorizing output 2nd 3rd 4th 5th 6th 7th
enabling application tracing 2nd 3rd 4th
enabling page tracing 2nd 3rd 4th
equipping pages for Trace mode 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
trace information categories 2nd
viewing application trace data from remote browser windows 2nd 3rd
Web.Config/Machine.Config Trace settings 2nd 3rd 4th
Windows event log 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
declaration
XML 2nd 3rd
delegation (controls) 2nd 3rd 4th
deleting
performance monitor categories 2nd 3rd
deleting files (XCOPY) 2nd 3rd
delimited data (XML) 2nd 3rd
enclosing character data 2nd 3rd 4th 5th
CDATA elements 2nd 3rd
character entities 2nd 3rd 4th
dependencies (cached items) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
deploying components 2nd
deployment 2nd 3rd 4th 5th
XCOPY 2nd 3rd
(begin)
(end)
confirmations/overwriting files 2nd 3rd
deploying directory trees 2nd 3rd 4th 5th
deploying only modified files 2nd 3rd
deploying single directories 2nd
excluding files 2nd 3rd 4th
switches 2nd 3rd 4th 5th 6th 7th 8th
Description attribute, Page object
Description property (WebMethodAttribute class) 2nd 3rd 4th 5th 6th 7th
directives, Page object 2nd 3rd 4th 5th
@Assembly
@Control
@Implements 2nd
@Import 2nd
@OutputCache
@Page 2nd 3rd 4th 5th 6th
@Register
directories
deployment
directory trees 2nd 3rd 4th 5th

This document is created with the unregistered version of CHM2PDF Pilot

excluding files 2nd 3rd 4th
single directories 2nd
DISCO files (XML)
Disposed event
Page object
Web controls
DLLs (dynamic link libraries) 2nd
Document Object Model (DOM)
XML DOM 2nd 3rd 4th 5th 6th 7th 8th 9th
extended classes 2nd
fundamental classes 2nd 3rd
Document Type Definitions (DTDs), XML 2nd 3rd 4th
documents
XML. [See XML]2nd [See XML]
DOM (Document Object Model)
XML DOM 2nd 3rd 4th 5th 6th 7th 8th 9th
extended classes 2nd
fundamental classes 2nd 3rd
dot (.) operator
XPath queries
DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
DTDs (Document Type Definitions), XML 2nd 3rd 4th
Duration attribute (OutputCache directive) 2nd
dynamic link libraries (DLLs) 2nd
dynamic reporting (HttpHandlers) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
SqlHandler 2nd 3rd 4th 5th 6th 7th
SqlHandler output
XSL template used to format output 2nd 3rd 4th 5th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

E switch (XCOPY) 2nd
e-mail
PreferredEmail attribute (PassportIdentity class)
sending errors logs to Web masters 2nd 3rd 4th 5th 6th
SmtpMail class 2nd 3rd
elements (XML) 2nd 3rd
. [See also nodes (XML)]
abbreviated close-tag syntax 2nd 3rd
case sensitivity 2nd
elements that contain data 2nd 3rd 4th
enclosing character data 2nd 3rd 4th 5th
CDATA elements 2nd 3rd
character entities 2nd 3rd 4th
listings
10.3, XML document with a declaration and a top-level element 2nd
10.4, XML document with elements that contain data 2nd
10.5, XML document with multiple child elements beneath the top-level element 2nd 3rd
10.6, XML document with elements and attributes 2nd
10.7, XML document with escape sequences 2nd
10.8, XML document with a CDATA section 2nd
10.9, XML document with a empty elements 2nd
slash (/) character
EnableSession property (WebMethodAttribute class) 2nd 3rd 4th 5th
EnableSessionState attribute, Page object
EnableViewState attribute, Page object 2nd
EnableViewStateMac attribute, Page object 2nd
EndRequest HttpModule 2nd 3rd 4th 5th
(time stamping output HTML)
enumerable classes 2nd 3rd
ChildNodes 2nd 3rd 4th 5th
Error event
Page object
error handling
ADO.NET 2nd 3rd 4th 5th 6th
customErrors settings (Web.Config/Machine.Config files) 2nd 3rd
database connections 2nd 3rd
ErrorPage attribute, Page object
errors
ADO.NET
handling errors 2nd 3rd 4th 5th 6th
SqlError class 2nd
validation error summaries 2nd 3rd 4th
customErrors settings (Web.Config/Machine.Config files) 2nd 3rd
debugging. [See debugging]
handling
HttpModule error module 2nd 3rd 4th 5th 6th 7th 8th 9th
sending errors logs to Web masters 2nd 3rd 4th 5th 6th
event handlers
Global.asax files 2nd 3rd 4th

This document is created with the unregistered version of CHM2PDF Pilot

event handling 2nd 3rd 4th 5th
AutoEventWireup attribute, Page object 2nd
IPostBackEventHandler
user controls 2nd 3rd
XML validation events 2nd 3rd
EventLog class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
events 2nd 3rd 4th 5th
. [See also classes]
Global.asax files 2nd 3rd 4th
Page object 2nd 3rd 4th 5th
raising via HttpModules 2nd 3rd 4th 5th 6th
server controls 2nd 3rd 4th
Web controls 2nd 3rd
Windows event log 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
XML validation 2nd 3rd
exceptions
ADO.NET SqlException class 2nd
EXCLUDE
switch (XCOPY)
exclude files 2nd 3rd
(XCOPY deployment)
EXCLUDE switch (XCOPY) 2nd
ExecuteScalar method 2nd 3rd 4th
(SQLCommand object) 2nd 3rd
executionTimeout section (Web.Config/Machine.Config files) 2nd
expiring cached items 2nd 3rd 4th 5th
removing expired items 2nd
extended XML objects 2nd
eXtensible Markup Language. [See XML]

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

F switch (XCOPY)
files
authorization 2nd
configuration files
accessing configuration files programmatically 2nd 3rd 4th
ASP.old versus ASP.NET 2nd
authentication settings 2nd 3rd 4th
authorization settings 2nd
browser capabilities settings 2nd
compilation settings 2nd 3rd
custom error settings 2nd 3rd
editing Web configuration files in Visual Studio.NET 2nd
execution timeout settings 2nd
global/local files 2nd
globalization settings 2nd 3rd
HTTP handler settings 2nd
HTTP module settings 2nd
locking down Web configuration settings 2nd
Machine.Config 2nd
multiple locations 2nd 3rd 4th 5th 6th
page settings 2nd 3rd
process model settings 2nd 3rd
section handlers
section handlers (Web.Config/Machine.Config files)
session state settings 2nd 3rd
trace settings
Trace settings
trace settings 2nd
Web services settings 2nd
Web.Config 2nd
Web.Config. [See also Web.Config files]
copying
XCOPY. [See XCOPY deploymnet]
deleting (XCOPY) 2nd 3rd
directories
. [See also directories]
overwriting (XCOPY) 2nd 3rd
uploading
HtmlInputFile control 2nd 3rd 4th 5th 6th 7th 8th
filtering output with HttpModules 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
forking filters 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
FirstChild() property (XML) 2nd
folders
. [See also directories]
forking output filters (HttpModules) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
form state 2nd 3rd
forms
. [See also Web Forms]
binding Web services 2nd 3rd 4th 5th

This document is created with the unregistered version of CHM2PDF Pilot

caching
creating copies for different browers
RequiredFieldValidation control 2nd 3rd
caching form results 2nd 3rd 4th 5th 6th
calling Web services from Web forms 2nd 3rd 4th 5th 6th 7th 8th 9th
calling Web services from Windows forms 2nd 3rd 4th 5th 6th 7th
grouping controls (Panel class) 2nd 3rd
HtmlForm control 2nd 3rd 4th 5th 6th 7th 8th 9th
Web Forms. [See Web Forms]
Windows Forms
Web services;creating cookie containers/utilizing State Web service 2nd 3rd 4th 5th 6th 7th 8th
forms authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
advantages 2nd
process flow 2nd 3rd 4th 5th
roles 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
settings 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
using forms authentication with URL authorization 2nd 3rd 4th
FormsAuthenticationTicket class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
versus Windows authentication 2nd 3rd
FormsAuthenticationTicket class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
handling the Application_AuthenticateRequest in global.asax 2nd 3rd 4th 5th
login.aspx HTML 2nd 3rd
class file 2nd 3rd 4th
properties
Transact SQL to create tables and stored procedures used by login.aspx 2nd 3rd 4th 5th 6th 7th
UserData member 2nd 3rd
UserRoleMappings table relationships 2nd
forward slash (/) character
XML closing tags
fundamental XML objects 2nd 3rd

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

GAC (Global Assembly Cache) 2nd 3rd 4th 5th
gacutil command-line utility 2nd 3rd
Gender attribute (PassportIdentity class)
GenericIdentity class 2nd 3rd
GenericPrincipal class 2nd
GenericPrincipal object 2nd 3rd 4th 5th 6th 7th 8th 9th
GetProfileObject method (PassportIdentity class) 2nd
Global Assembly Cache (GAC) 2nd 3rd 4th 5th
gacutil utility 2nd 3rd
global.asax
Application_AuthenticateRequest handler 2nd 3rd 4th
Global.asax
raising events via HttpModules 2nd 3rd 4th
global.asax
redirecting users from error pages via HttpModules 2nd 3rd
global.asax class
forms authentication 2nd 3rd
Global.asax files 2nd
(begin)
(end)
application directives 2nd 3rd 4th
code declaration blocks 2nd 3rd
listings
6.2, OnStart event handler in Global.asax 2nd 3rd
6.3, declaring Application-Scoped objects 2nd 3rd
6.4, displaying results of Active-User array 2nd 3rd
server-side includes 2nd
server-side object tags 2nd 3rd 4th 5th
globalization section (Web.Config/Machine.Config files) 2nd 3rd
graphics
Image control 2nd 3rd
ImageButton control 2nd 3rd 4th
grouping Web forms controls (Panel class) 2nd 3rd
groups

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

H switch (XCOPY)
handling errors
ADO.NET 2nd 3rd 4th 5th 6th
customErrors settings (Web.Config/Machine.Config files) 2nd 3rd
database connections 2nd 3rd
HttpModule error module 2nd 3rd 4th 5th 6th 7th 8th 9th
handling events 2nd 3rd 4th 5th
AutoEventWireup attribute, Page object 2nd
Global.asax files 2nd 3rd 4th
IPostBackEventHandler
raising events via HttpModules 2nd 3rd 4th 5th 6th
user controls 2nd 3rd
XML validation 2nd 3rd
handling exceptions
ADO.NET SqlException class 2nd
hard-coded impersonation 2nd 3rd 4th 5th 6th 7th
hash tables
using Repeater control to build customized hashtable output 2nd 3rd
hashtables
listing 2.33, DataList server control bound to a Hashtable object 2nd 3rd
headers
HttpCacheVaryByHeaders class 2nd
Set-Cookie 2nd 3rd
VaryByHeader caching 2nd 3rd
Headers attribute (OutputCache directive) 2nd
Hello World
HttpHandlers 2nd 3rd 4th 5th 6th 7th
code listing 2nd 3rd
creating 2nd 3rd
Hello World server control 2nd 3rd
ASP.NET page that references custom Hello World server control 2nd 3rd 4th
batch file that compiles and deploys a .NET component project automatically 2nd
compiling namespaces/classes into .NET components
HelloWorld Web service 2nd 3rd 4th
hidden text
HtmlInputHidden control 2nd 3rd 4th 5th 6th 7th 8th
hit counters (HttpHandlers) 2nd 3rd 4th 5th 6th 7th 8th
HTML
HTML data representation versus XML
HTML 2nd 3rd 4th 5th 6th 7th 8th 9th
XML 2nd 3rd
tags
HTML control equivalents 2nd 3rd
Web control equivalents 2nd 3rd
time stamping output HTML
(BeginRequest-EndRequest module) 2nd 3rd 4th 5th
View Source of a Page That Has Been Affected by the BeginEnd Module
HTML controls 2nd
equivalent HTML tags 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

HtmlAnchor 2nd 3rd 4th 5th
HtmlButton 2nd 3rd 4th 5th
HtmlForm 2nd 3rd 4th 5th 6th 7th 8th 9th
HtmlImage 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
HtmlInputButton 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputCheckbox 2nd
HtmlInputFile 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputHidden 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputImage 2nd
HtmlInputRadioButton 2nd
HtmlInputText 2nd
HtmlSelect 2nd
HtmlTable 2nd
HtmlTableCell 2nd
HtmlTableCellCollection 2nd
HtmlTableRow 2nd
HtmlTableRowCollection 2nd
HtmlTextArea 2nd
HtmlAnchor control 2nd 3rd 4th 5th
HtmlButton control 2nd 3rd 4th 5th
HtmlForm control 2nd 3rd 4th 5th 6th 7th 8th 9th
HtmlImage control 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
HtmlInputButton control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputCheckbox control 2nd
HtmlInputFile control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputHidden control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputImage control 2nd
HtmlInputRadioButton control 2nd
HtmlInputText control 2nd
HtmlSelect control 2nd
HtmlTable control 2nd
HtmlTableCell control 2nd
HtmlTableCellCollection control 2nd
HtmlTableRow control 2nd
HtmlTableRowCollection control 2nd
HtmlTextArea control 2nd
HTTP
Telnet requests 2nd 3rd 4th 5th
HTTP GET
calling Web method (Web services) 2nd
HTTP handlers
trace.axd 2nd 3rd
HttpApplication class 2nd 3rd 4th 5th 6th 7th 8th
HttpBrowserCapabilities class 2nd 3rd 4th 5th 6th 7th
caching 2nd 3rd 4th
HttpCachePolicy class 2nd
HttpCacheVaryByHeaders class 2nd
HttpCacheVaryByParams class 2nd
HttpForbiddenHandler HttpHandler
. [See also HttpHandlers]
HttpHandler
listings

This document is created with the unregistered version of CHM2PDF Pilot

8.17, HttpHandlerFactory that returns different handlers based on the name parameter 2nd 3rd
HttpHandlers 2nd 3rd 4th 5th 6th 7th 8th 9th
(begin)
(end)
ASP.NET request handling 2nd 3rd
built-in ASP.NET HttpHandlers 2nd
combining XML/SQL 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
SqlHandler 2nd 3rd 4th 5th 6th 7th
SqlHandler output
XSL template used to format output 2nd 3rd 4th 5th
dynamic handler assignment 2nd 3rd 4th 5th 6th
Hello World 2nd 3rd 4th 5th 6th 7th
code listing 2nd 3rd
creating 2nd 3rd
IHttpHandler class 2nd
listings
8.10, global.asax that sinks the OnMyEvent event from the HttpModule 2nd
8.13, Hello World 2nd 3rd
8.14, SqlHandler transforms XML SQL queries from SQL Server with XSL templates 2nd 3rd 4th 5th
8.15, XSL template used to format output 2nd 3rd
8.16, page counter 2nd 3rd 4th
8.9, Handler That Raises an Event in Global.asax 2nd 3rd
page counters 2nd 3rd 4th 5th 6th 7th 8th
Web.Config/Machine.Config httpHandlers settings
HttpModules 2nd 3rd
(begin)
(end) 2nd
ASP.NET request handling 2nd 3rd
authentication modules 2nd 3rd 4th
BeginRequest-EndRequest module 2nd 3rd 4th 5th
(time stamping output HTML)
error modules 2nd 3rd 4th 5th 6th 7th 8th 9th
filtering output 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
forking filters 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
IhttpModule class
listings
8.1, Module That Stamps the Begin and End Times into the Page 2nd 3rd 4th
8.11, authenticateRequest in a Web module 2nd 3rd
8.12, RewritePath module that changes every request to map to webform1.aspx 2nd 3rd
8.2, web.config to Load the BeginEnd HttpModule 2nd
8.3, View Source of a Page That Has Been Affected by the BeginEnd Module 2nd 3rd
8.4, simple ad insertion engine that replaces <adinsert tags with an ad 2nd 3rd 4th
8.5, output from AdInserter.vb 2nd 3rd
8.6, SimpleCache, an Output-Caching Mechanism 2nd 3rd 4th 5th 6th 7th
8.7, module that handles errors in an application by writing them to the event log and e-mailing the 2nd 3rd 4th
8.8, Global.asax redirection of users from error pages 2nd 3rd
raising events 2nd 3rd 4th 5th 6th
rewriting paths 2nd 3rd 4th 5th 6th
Web.Config/Machine.Config httpModules settings 2nd
HttpRequest class 2nd
HttpResponse class 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

HttpServerUtility class 2nd
HttpSessionState class 2nd 3rd 4th 5th
. [See also state management]
TimeOut property 2nd
Hyperlink class 2nd 3rd
hyperlinks
hyperlink control that can generate client-side postback of a form 2nd 3rd
LinkButton control 2nd 3rd

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

I switch (XCOPY)
identities 2nd
  GenericIdentity class 2nd 3rd
  impersonation 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
    determining identity with WindowsIdentity 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
    hard-coded impersonation 2nd 3rd 4th 5th 6th 7th
  PassportIdentity class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
    attributes 2nd
  WindowsIdentity class 2nd
IEnumerable interface 2nd 3rd
  ChildNodes 2nd 3rd 4th 5th
IHttpHandler class 2nd
  . [See also HttpHandlers]2nd [See also HttpHandlers]
IhttpModule class 2nd
  . [See also HttpModules]
IIS
  authorization
Image class 2nd 3rd
image.aspx listing 2nd 3rd 4th
ImageButton class 2nd 3rd 4th
images
  HtmlImage control 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
  HtmlInputImage control 2nd
impersonation 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
  determining identity with WindowsIdentity 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
  hard-coded impersonation 2nd 3rd 4th 5th 6th 7th
implementing interfaces
  @Implements directive (Page object) 2nd
importing namespaces
  @Import directive (Page object) 2nd
in process session state 2nd 3rd 4th
  advantages/disadvantages 2nd 3rd 4th 5th 6th 7th 8th 9th
inequality operator (!=), XPath queries
Inherits attribute, Page object
Init event
  Page object
  Web controls
Init event (custom server controls)
initializing applications using Global.asax files 2nd
  (begin)
  (end)
  application directives 2nd 3rd 4th
  code declaration blocks 2nd 3rd
  server-side includes 2nd
  server-side object tags 2nd 3rd 4th 5th
initializing user states 2nd
  . [See also state management]
inline method of passing parameters
input

This document is created with the unregistered version of CHM2PDF Pilot

buttons. [See buttons]
check boxes
HtmlInputCheckbox control 2nd
HtmlInputFile control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputHidden control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputImage control 2nd
text
. [See also text]2nd [See also text]
HtmlInputText control 2nd
inputbutton.aspx listing 2nd 3rd 4th 5th 6th
inputfile.aspx listing 2nd 3rd 4th 5th 6th
Insert argument (Cache class)
InsertAfter method (XmlNode object) 2nd 3rd
Inside.aspx code listing 2nd
intercases
Web controls. [See Web controls]
interfaces
implementing
@Implements directive (Page object) 2nd
Internet Explorer
rich clients 2nd
Smart Navigation features
IPostBackEventHandler
IPrincipal interface 2nd
ISAPI filter
security
ItemPriority argument (Cache class)

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

K switch (XCOPY)
Keep-Alive (HTTP 1.1)

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

L switch (XCOPY)
Label class 2nd 3rd
langagues
globalization section (Web.Config/Machine.Config files) 2nd 3rd
language
Lang_Preference attribute (PassportIdentity class)
Language attribute, Page object
languages
CodePage attribute 2nd
Common Language Specification (CLS) 2nd
Language attribute (Page object)
LastFiveSearchTerms code listing 2nd 3rd
LCID attribute, Page object 2nd
LinkButton class 2nd 3rd
links
. [See also hyperlinks]
ListBox class 2nd 3rd
listing
pages
View State 2nd 3rd
listing components in GAC 2nd 3rd 4th
listings
AdRotator class 2nd 3rd 4th
caching
browser capabilities 2nd
containing page defining cache directive 2nd
containing page without cache directive 2nd
correct form output caching 2nd 3rd 4th
creating cached page for each header 2nd
directive for 5-minute cache 2nd
expiring items 2nd 3rd
form creating copies for different browsers 2nd 3rd
form using RequiredFieldValidator 2nd
incorrect form output caching 2nd 3rd 4th
partial page caching of previous search terms (LastFiveSearchTerms) 2nd 3rd
partial page caching user control (SalesByEmployee.ascx) 2nd 3rd 4th 5th
partial page caching user control (SalesPeople.ascx) 2nd 3rd
query that places output into drop-downlist 2nd 3rd
query that stores output in cache 2nd 3rd 4th
re-adding dropped out cache items 2nd 3rd 4th
refreshing cached items via dependencies 2nd 3rd
removing/inserting items from cache with expirations 2nd 3rd 4th 5th
search page containing term caching (BreadCrumbSearch.aspx) 2nd 3rd 4th
Calendar server control 2nd
CheckBox server control 2nd 3rd
databases
binding DataGrid control to ArrayList object 2nd
calling a parameterized stored procedure 2nd 3rd 4th
calling a parameterized stored procedure using SqlCommand Parameters collection 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

calling a simple stored procedure using SqlCommand 2nd 3rd
comparison validation 2nd 3rd
custom validation 2nd 3rd
data-entry form 2nd 3rd 4th
displaying data access errors 2nd 3rd
displaying query data in Web browsers 2nd 3rd
displaying query data in Web Forms DataGrid control 2nd 3rd
displaying query results using SqlDataAdapter and DataSet 2nd 3rd
displaying validation error summary 2nd 3rd 4th
ExecuteScalar method 2nd
executing a text command that returns a SqlDataReader object 2nd 3rd
executing nonquery commands 2nd 3rd
opening a connection using OLE DB provider 2nd
opening a connection using SQL Server provider 2nd 3rd
opening a connection with error handling 2nd 3rd
range validation 2nd 3rd
regular expression validation 2nd 3rd
required field funcion, client-side validation 2nd
required field validation 2nd 3rd
using Errors collection to display rich error information 2nd
DataList server control
DataList server control bound to a Hashtable object 2nd 3rd
debugging
calling Debug.Write to display debug information in development environment 2nd
test page for debugging 2nd 3rd
Web.config file that activates application-level debugging 2nd
DropDownList control, binding to data in ArrayList object 2nd 3rd
Global.asax files
declaring Application-Scoped objects 2nd 3rd
displaying results of Active-User array 2nd 3rd
Global.asax OnStart event handler 2nd 3rd
HttpHandler
HttpHandlerFactory that returns different handlers based on the name parameter 2nd 3rd
HttpHandlers
global.asax that sinks the OnMyEvent event from the HttpModule 2nd
Handler That Raises an Event in Global.asax 2nd 3rd
Hello World 2nd 3rd
page counter 2nd 3rd 4th
SqlHandler transforms XML SQL queries from SQL Server with XSL templates 2nd 3rd 4th 5th
XSL template used to format output 2nd 3rd
HttpModules
authenticateRequest in a Web module 2nd 3rd
Global.asax redirection of users from error pages 2nd 3rd
module that handles errors in an application by writing them to the event log and e-mailing the Webm 2nd 3rd
Module That Stamps the Begin and End Times into the Page 2nd 3rd 4th
RewritePath module that changes every request to map to webform1.aspx 2nd 3rd
simple ad insertion engine that replaces <adinsert tags with an ad 2nd 3rd 4th
SimpleCache, an Output-Caching Mechanism 2nd 3rd 4th 5th 6th 7th
View Source of a Page That Has Been Affected by the BeginEnd Module 2nd 3rd
web.config to Load the BeginEnd HttpModule 2nd
logging
writing events to Windows event log 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

pages
@Page directive 2nd
2.10, HtmlAnchor control HTML 2nd 3rd
2.24, code-behind Page_Load event procedure without AutoEventWireup 2nd
Button.aspx code 2nd 3rd 4th
Button.aspx HTML 2nd
Calendar.aspx 2nd
Calendar.aspx rendered HTML source 2nd 3rd 4th
Form.aspx code 2nd 3rd 4th
Form.aspx HTML 2nd
HtmlAnchor control code 2nd
HtmlInputHidden control 2nd 3rd
HttpBrowserCapabilities in Request object 2nd 3rd
image.aspx code 2nd 3rd 4th
image.aspx HTML 2nd 3rd 4th
inputbutton.aspx code 2nd 3rd 4th
inputbutton.aspx HTML 2nd 3rd
inputfile.aspx code 2nd 3rd 4th
inputfile.aspx HTML 2nd
SamplePage2.aspx, event handler for a button 2nd
SamplePage3.aspx, using code behind 2nd 3rd
SimplePage.asp 2nd 3rd
SimplePage.aspx, a reworking of SimplePage.asp in ASP.NET 2nd 3rd
SimplePage2.asp, showing code to preserve form state in ASP.OLD 2nd 3rd
SimplePage3.asp, reaction to user interaction in ASP.old 2nd 3rd
SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd 3rd
performance monitors
deleting a performance category 2nd
incrementing performance monitor using PerformanceCounter object 2nd 3rd 4th
initializing new monitor category and monitor object 2nd
Repeater control, building customized hashtable output 2nd 3rd
security
global.asax class that handles the AuthorizeRequest application event 2nd 3rd
global.asax containing the Application_AuthenticateRequest handler 2nd 3rd
HTML for login.aspx 2nd 3rd
HTML for login.aspx, class file 2nd 3rd 4th 5th
login page that authenticates only two users 2nd 3rd 4th
login page that authenticates only two users, class file 2nd 3rd 4th
login.aspx page shown when users attempt to access content when they are not authenticated 2nd 3rd
page that displays the PUID, e-mail address, and member name of a logged-in Passport user 2nd 3rd
simple login page 2nd 3rd 4th 5th
simple login page, class file 2nd 3rd 4th 5th 6th
Transact SQL to create tables and stored procedures used by login.aspx 2nd 3rd 4th 5th
user authentication with WindowsPrincipal 2nd 3rd 4th
user authentication with WindowsPrincipal, class file 2nd
Web User Control That Wraps the Display of the Passport Logo 2nd 3rd 4th
web.config file setting authentication to forms and defining URL authorization 2nd
web.config file that uses URL authorization Location element 2nd 3rd 4th
Web.config for simple forms authentication 2nd 3rd
Web.config set up for impersonation 2nd 3rd
web.config setup for Passport authentication 2nd
Web.config that maps roles to application locations 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

Web.config with a hard-coded impersonation account 2nd 3rd
Webform1.aspx 2nd 3rd 4th 5th
Webform1.aspx, code behind class 2nd 3rd 4th 5th
server controls
ASP.NET page that references custom Hello World server control 2nd 3rd
ASP.NET Page that uses an instance of subclassed CustomText control 2nd 3rd
basic control that stores property state 2nd
batch file that compiles and deploys a .NET component project automatically 2nd
composite control 2nd 3rd
CustomTextBox control that contains set of custom formatting defaults 2nd
Hello World server control example 2nd 3rd
hyperlink control that can generate client-side postback of a form 2nd 3rd
namespace and class that can be compiled into a .NET component 2nd
page created to host an instance of composite Search server control 2nd
page to contain counter control 2nd 3rd
rewritten CurrentValue property, demonstrating lack of state persistence 2nd
using delegation to provide access to Text property of child control 2nd 3rd
state management
adding/displaying session values 2nd 3rd
getting a session value 2nd
HTTP request using Telnet 2nd 3rd
out of process session state 2nd
relative and absolute references 2nd 3rd
Session Start event 2nd
Set-Cookie header 2nd
setting a session value 2nd
starting State Server 2nd
timing read/write operations on session state 2nd 3rd 4th
tracing
using Web.config to activate tracing for entire Web directories 2nd
categorized Trace.Write output 2nd 3rd
simple page equipped for tracing with calls to Trace.Write 2nd 3rd 4th
user controls
custom control utilized in an ASP.NET Web Form 2nd
handling events 2nd 3rd
Register directive referencing a user control 2nd
Search control with ShowAdvanced method 2nd 3rd
Search dialog box user control 2nd 3rd
Search dialog box with TitleBarText property 2nd 3rd
Web controls
Web form declaration
Web services
Activator .asmx file 2nd
Code Behind Web Service Class That Implements a Cached Time Service 2nd
code-behind file for simple state service 2nd
Description property 2nd
Form That Is Bound to the SimpleDataSet Web Service 2nd 3rd 4th 5th
HelloWorld 2nd
Proxy Class (datasetsample.vb) for SimpleDataSet Generated with the WSDL Tool 2nd 3rd
Proxy Created for SimpleDataSet by Visual Studio .NET 2nd 3rd 4th
sample invoice in HTML 2nd 3rd 4th 5th 6th
sample invoice in XML 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

service that returns ArrayList 2nd 3rd
Simple() WebMethod of SimpleDataSet 2nd 3rd 4th
Web Form Client for SimpleDataSet 2nd
Web Form That Calls the NorthwindOrder Web Service Asynchronously 2nd 3rd 4th 5th
Web Form That Calls the NorthwindOrders Service Asynchronously and Loads the Orders into a Grid 2nd 3rd 4th
Web Method That Returns a Dataset with a Master Detail Relationship 2nd 3rd
Windows Form That Creates a Cookie Container and Utilizes the State Web Service 2nd 3rd 4th 5th 6th
XML Output from Calling the GetOrders WebMethod with a Date of 7/8/1996 2nd 3rd 4th 5th 6th 7th
Web.Config file with Location section to handle subdirectory settings 2nd 3rd
XCOPY exclude file 2nd
XML
W3C schema containing a complex type that refers to a simple type 2nd
books.xml document 2nd 3rd
books.xml File Containing a Link to the books.xsl Style Sheet 2nd
booksellerÕs hypothetical output 2nd
ChildNodes class 2nd 3rd 4th
comma-delimited document 2nd
inserting new child nodes using AppendChild method 2nd
loading a local XML file using XmlDocument .Load() method 2nd
loading an XML file residing on a Web server 2nd
programmatically transforming an XML file using an XSL style sheet and the XslTransform object 2nd
publisherÕ
publisherÕs hypothetical input
responding to events raised by Validate subroutine 2nd 3rd
simplified book.xml document 2nd
Validation subroutine using XmlValidatingReader object 2nd 3rd
W3C schema containing a definition for a simple type 2nd
W3C schema containing an element definition that refers to a type definition 2nd
W3C schema definition boilerplate 2nd
XDR Schema 2nd
XML 1.0 declaration
XML document with a CDATA section 2nd
XML document with a declaration and top-level element 2nd
XML document with a empty elements
XML document with elements and attributes 2nd
XML document with elements that contain data 2nd
XML document with escape sequences 2nd
XML document with multiple child elements beneath the top-level element 2nd 3rd
XmlNode object InsertAfter method 2nd 3rd
XmlNodeReader object, creating from XmlDocument object 2nd
XmlNodeReader object, traversing a document 2nd
XmlTextReader object 2nd 3rd
XmlTextWriter object 2nd 3rd
XmlTextWriter object, document produced by XmlTextWriter object 2nd
XPath query combining multiple criteria with AND and OR 2nd
XPath query page used to test XPath queries against books.xml 2nd
XPath query results 2nd
XPath query that excludes nodes based on attribute values 2nd
XPath query that retrieves a specific BOOK node by querying on text in BookÕs TITLE node 2nd
XPath query that retrieves a specific TITLE node by querying on its text 2nd
XPath query to retrieve author information based on author ID 2nd

This document is created with the unregistered version of CHM2PDF Pilot

XPath query to retrieve multiple instances of same author 2nd 3rd
XPathNavigator object Select method 2nd
XpathNodeIterator Current property 2nd 3rd
XSLT Style Sheet That Converts an XML Document into an HTML Document 2nd 3rd
lists
DataList class 2nd 3rd 4th 5th 6th
DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
ListBox control 2nd 3rd
Load event
Page object
Web controls
Load event (custom server controls)
Load() method (XML) 2nd 3rd 4th 5th 6th
listings
10.11, loading a local XML file
10.12, loading an XML file residing on a Web server
10.13, loading a local XML file 2nd
LoadPostData event (custom server controls)
LoadViewState event (custom server controls)
locale identifier (LCID)
LCID attribute (Page object) 2nd
Location attribute (OutputCache directive) 2nd 3rd 4th
location element, URL authorization; 2nd 3rd 4th 5th
location section (Web.Config) 2nd 3rd 4th 5th 6th
locking down Web configuration settings 2nd
logging
listings
3.10, writing events to Windows event log 2nd 3rd
Windows event log 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
logic
business logic componenets
Global Assembly Cache (GAC) 2nd 3rd 4th
logic, separating code from presentation using code behind 2nd 3rd 4th 5th 6th 7th 8th
. [See also code behind]
listing 2.8, SamplePage3.aspx, using code behind 2nd 3rd
listing 2.9, SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd
login page
forms authentication 2nd 3rd 4th 5th 6th 7th
login pages
authenticating only two users 2nd 3rd 4th
class file 2nd 3rd 4th
HTML for login.aspx 2nd 3rd
class file 2nd 3rd 4th 5th
Passport authentication 2nd 3rd
Transact SQL to create tables and stored procedures used by login.aspx 2nd 3rd 4th 5th
loginurl attribute
forms authentication
LogoTag2( ) method (Passport authentication)
logs
EventLog class 2nd 3rd 4th
HttpModule That Handles Errors in an Application by Writing Them to the Event Log and E-mailing the
HttpModule that handles errors in an application by writing them to the event log and e-mailing the 2nd 3rd 4th

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

M switch (XCOPY)
Machine.Config file 2nd
  accessing configuration files programmatically 2nd 3rd 4th
  editing Web configuration files in Visual Studio.NET 2nd
  locking down Web configuration settings 2nd
  section handlers 2nd
    authentication 2nd 3rd 4th
    authorization 2nd
    browser capabilities 2nd
    compilation 2nd 3rd
    custom errors 2nd 3rd
    execution timeout 2nd
    globalization 2nd 3rd
    HTTP handlers
    HTTP modules 2nd
    pages 2nd 3rd
    process model 2nd 3rd
    session state 2nd 3rd
    trace 2nd 3rd 4th
    Web services 2nd 3rd
Master-Detail relationships
  Web service DataSets 2nd 3rd 4th 5th 6th 7th 8th 9th
  Web service DataSets (should be serialization) 2nd
  Web service serialization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
MaxItems argument (Cache class) 2nd
MemberName attribute (PassportIdentity class)
methods
  . [See also classes]
  AdRotator class
  Button class
  Cache class
  CacheDependency class
  Calendar class 2nd
  CheckBox class
  Control class
  DataAdapter class (ADO.NET)
  DataColumn class (ADO.NET)
  DataGrid class 2nd
  DataList class 2nd
  DataRelation class (ADO.NET)
  DataSet class (ADO.NET)
  DataTable class (ADO.NET)
  Debug class
  DropDownList class
  EventLog class
  HtmlAnchor control
  HtmlButton control
  HtmlForm control
  HtmlImage control

This document is created with the unregistered version of CHM2PDF Pilot

HtmlInputButton control
HtmlInputCheckBox control
HtmlInputFile control
HtmlInputHidden control
HtmlInputImage control
HtmlInputRadioButton control
HtmlInputText control
HtmlSelect control
HtmlTable control
HtmlTableCell control
HtmlTableCellCollection control
HtmlTableRow control
HtmlTableRowCollection control
HtmlTextArea control
HttpApplication class
HttpCachePolicy class 2nd
HttpCacheVaryByHeaders class
HttpRequest class
HttpResponse class
HttpServerUtility class
HttpSessionState class 2nd
Hyperlink class 2nd
Image class 2nd
ImageButton class
Label class 2nd
LinkButton class 2nd
ListBox class
OleDbCommand class (ADO.NET)
Page class
Panel class 2nd
PerformanceCounter class
RadioButton class 2nd
Repeater class
SqlConnectionclass (ADO.NET)
SqlDataAdapter class (ADO.NET)
SqlDataReader class (ADO.NET)
SqlError (ADO.NET)
SqlException (ADO.NET)
Table class 2nd
TableCell class 2nd
TableRow class
TextBox class
TraceContext class
user controls 2nd 3rd 4th 5th
ValidationEventArgs class
WebControl class 2nd
XmlDocument class
XmlNamedNodeMap class
XmlNode class
XmlNodeList class
XmlNodeReader class 2nd
XmlReader class

This document is created with the unregistered version of CHM2PDF Pilot

XmlTextReader class
XmlTextWriter class
XmlValidatingReader class
XmlWriter class
Microsoft
XML Web site
Microsoft Knowledge Base Web site
Windows 2000 command changes
Microsoft Passport. [See Passport authentication]
Microsoft Web site
XDR schemas
mobile Web forms 2nd
monitoring applications
performance monitors. [See performance monitors]2nd [See performance monitors]3rd [See performance monitors]4th [See performance monitors]5th [See performance monitors]6th [See performance monitors]7th [See performance monitors]8th [See performance monitors]9th [See performance monitors]10th [See performance monitors]11th [See performance monitors]12th [See performance monitors]13th [See performance monitors]14th [See performance monitors]15th [See performance monitors]
tracing 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th
categorizing output 2nd 3rd 4th 5th 6th 7th
enabling application tracing 2nd 3rd 4th
enabling page tracing 2nd 3rd 4th
equipping pages for Trace mode 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
trace information categories 2nd
viewing application trace data from remote browser windows 2nd 3rd
Web.Config/Machine.Config Trace settings 2nd 3rd 4th
Windows event log 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
MOVE command
Windows 2000 changes

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

N switch (XCOPY)
name attribute
forms authentication
namespaces
compiling into .NET components 2nd
importing
@Import directive (Page object) 2nd
navigating XML documents 2nd 3rd
networks
data representation. [See data representation]
Nickname attribute (PassportIdentity class)
nodes
XML
manipulating current nodes using XPathNodeIterator Current property 2nd 3rd
nodes (XML) 2nd
abbreviated close-tag syntax 2nd 3rd
case sensitivity 2nd
ChildNodes class 2nd 3rd 4th 5th
complex/simple types 2nd 3rd 4th 5th 6th 7th
creating attributes 2nd
elements that contain data 2nd 3rd 4th
enclosing character data 2nd 3rd 4th 5th
CDATA elements 2nd 3rd
character entities 2nd 3rd 4th
inserting new child nodes using AppendChild method 2nd
listings
10.3, XML document with a declaration and a top-level element 2nd
10.4, XML document with elements that contain data 2nd
10.5, XML document with multiple child elements beneath the top-level element 2nd 3rd
10.6, XML document with elements and attributes 2nd
10.7, XML document with escape sequences 2nd
10.8, XML document with a CDATA section 2nd
10.9, XML document with empty elements 2nd
slash (/) character
XmlNode class 2nd 3rd 4th
InsertAfter method 2nd 3rd
viewing document data 2nd 3rd 4th 5th 6th 7th 8th
XmlNodeList class 2nd
XmlNodeReader class 2nd 3rd 4th
navigating/updating documents 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
nonquery commands (ADO.NET) 2nd 3rd 4th
Northwind database
calling NorthwindOrders Web service asynchronously via Web Form 2nd 3rd 4th 5th
Northwind Database
SalesByCategory stored procedure
passing parameters to stored procedures 2nd 3rd 4th 5th
Northwind database
Web service serialization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

O switch (XCOPY)
objects
. [See also classes]2nd [See also classes]3rd [See also classes]
command objects (ADO.NET) 2nd
data-containing objects in ADO.NET 2nd
database connections 2nd 3rd 4th 5th 6th 7th
Debug 2nd 3rd 4th
. [See also debugging]2nd [See also debugging]3rd [See also debugging]
identities 2nd 3rd 4th
PerformanceCounter 2nd 3rd 4th 5th
. [See also performance counters]
principal objects 2nd 3rd 4th
serialization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th
DataSet 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
server-side object tags
Global.asax files 2nd 3rd 4th 5th
Trace. [See tracing]
XML Document Object Model (DOM) 2nd 3rd 4th 5th 6th 7th 8th 9th
extended classes 2nd
fundamental classes 2nd 3rd
objectsApplication-Scoped objects, declaring in Global.asax files 2nd 3rd
OLE DB
connecting to databases 2nd 3rd 4th
OleDbAdapter object
. [See also data adapters (ADO.NET)]
OleDbCommand class 2nd 3rd
OleDbCommand class (ADO.NET) 2nd
OleDbConnection class 2nd 3rd 4th 5th 6th
OleDbConnection object 2nd 3rd 4th
OleDbDataReader class
OnStart event handler in Global.asax 2nd 3rd
Open method
database connections 2nd 3rd
. [See also connecting to databases]
Opera Web browser
operators
ValidationCompareOperator 2nd 3rd
OR keyword
XPath queries 2nd
out of process session state (State Server) 2nd 3rd 4th 5th 6th 7th
advantages/disadvantages 2nd 3rd 4th 5th 6th 7th 8th 9th
output
ad insertion engine that replaces <adinsert tags with ads 2nd 3rd
filtering with HttpModules 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
forking filters 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
form results
caching 2nd 3rd 4th 5th 6th
time stamping with HttpModules 2nd 3rd 4th 5th 6th 7th 8th
WSDL, Web services

This document is created with the unregistered version of CHM2PDF Pilot

output caching
. [See also caching]
output caching (entire pages) 2nd 3rd 4th 5th 6th 7th
OutputCache directive 2nd
. [See also caching]
caching partial pages 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
generating crumb trails of previous search terms 2nd 3rd 4th 5th
generating phone lists 2nd 3rd 4th 5th 6th
Duration attribute 2nd
Headers attribute 2nd
Location attribute 2nd 3rd 4th
partial pages
allowed user control/ output caching combinations 2nd 3rd 4th 5th
VaryByCustom attribute 2nd 3rd 4th 5th 6th
VaryByHeader attribute 2nd 3rd 4th
VaryByParam attribute 2nd 3rd 4th 5th 6th 7th
VaryByParams attribute 2nd
overwriting files (XCOPY) 2nd 3rd

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

P switch (XCOPY)
Page class 2nd 3rd
page counters (HttpHandlers) 2nd 3rd 4th 5th 6th 7th 8th
Page object
attributes 2nd 3rd
AspCompat 2nd
AutoEventWireup 2nd
Buffer 2nd
ClassName
ClientTarget 2nd
CodePage 2nd
CompilerOptions 2nd
ContentType 2nd
Culture 2nd
Debug
Description
EnableSessionState
EnableViewState 2nd
EnableViewStateMac 2nd
ErrorPage
Inherits
Language
LCID 2nd
SmartNavigation
Src 2nd
Trace
TraceMode
Transaction
UICulture
WarningLevel 2nd
directives 2nd 3rd 4th 5th
. [See also Page object, attributes]
@Assembly
@Control
@Implements 2nd
@Import 2nd
@OutputCache
@Page 2nd 3rd 4th 5th 6th
@Register
events 2nd 3rd 4th 5th
Page class 2nd 3rd
PageHandlerFactory HttpHandler
. [See also HttpHandlers]
pages
advertisements
rotating (AdRotator class) 2nd 3rd 4th 5th 6th
ASP.NET control model 2nd 3rd
(begin)
(end)

This document is created with the unregistered version of CHM2PDF Pilot

event handling 2nd 3rd 4th 5th 6th 7th
event handling. [See also event handling]
form state 2nd 3rd
HTML controls 2nd
HTML controls. [See also HTML controls]
server controls 2nd 3rd 4th 5th
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
buffering
deactivating via Buffer attribute 2nd
buttons
HtmlButton control 2nd 3rd 4th 5th
HtmlInputButton control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputRadioButton control 2nd
RadioButton control 2nd 3rd 4th
caching 2nd 3rd 4th 5th 6th 7th
. [See also caching]
@OutputCache directive (Page object)
browser differences 2nd 3rd 4th
partial pages 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
refreshing cached items via dependencies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
VaryByCustom/VaryByHeader 2nd 3rd
check boxes 2nd 3rd 4th
CheckBox control 2nd 3rd
debugging 2nd 3rd 4th
. [See also debugging]
determining browser capabilities 2nd 3rd 4th 5th 6th 7th
event handling 2nd 3rd 4th 5th
headers
HttpCacheVaryByHeaders class 2nd
Set-Cookie 2nd
VaryByHeader caching 2nd 3rd
HTML controls. [See HTML controls]2nd [See HTML controls]
HttpApplication class 2nd 3rd
HttpBrowserCapabilities class 2nd 3rd 4th 5th 6th 7th
HttpRequest class 2nd
HttpResponse class 2nd 3rd
HttpServerUtility class 2nd
HttpSessionState class 2nd 3rd
Hyperlink class 2nd 3rd
Image control 2nd 3rd
ImageButton control 2nd 3rd 4th
LinkButton control 2nd 3rd
ListBox control 2nd 3rd
listings
2.1, SimplePage.asp 2nd 3rd
2.11, HtmlAnchor control code 2nd
2.12, Button.aspx HTML 2nd
2.13, Button.aspx code 2nd 3rd 4th
2.14, Form.aspx HTML 2nd
2.15, Form.aspx code 2nd 3rd 4th
2.16, image.aspx HTML 2nd 3rd 4th
2.17, image.aspx code 2nd 3rd 4th

This document is created with the unregistered version of CHM2PDF Pilot

2.18, inputbutton.aspx HTML 2nd 3rd
2.19, inputbutton.aspx code 2nd 3rd 4th
2.2, SimplePage2.asp, showing code to preserve form state in ASP.OLD 2nd 3rd
2.20, inputfile.aspx HTML 2nd
2.21, inputfile.aspx code 2nd 3rd 4th
2.22, HtmlInputHidden control 2nd 3rd
2.23, @Page directive 2nd
2.26, View State 2nd 3rd
2.27, HttpBrowserCapabilities in Request object 2nd 3rd
2.3, SimplePage.aspx, a reworking of SimplePage.asp in ASP.NET 2nd 3rd
2.4, Calendar.aspx 2nd
2.5, Calendar.aspx rendered HTML source 2nd 3rd 4th
2.6, SimplePage3.asp, reaction to user interaction in ASP.old 2nd 3rd
2.7, SamplePage2.aspx, event handler for a button 2nd
2.8, SamplePage3.aspx, using code behind 2nd 3rd
2.9, SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd 3rd
code-behind Page_Load event procedure without AutoEventWireup 2nd
HtmlAnchor control HTML 2nd 3rd
lists
DataList class 2nd 3rd 4th 5th 6th
DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
Page class 2nd 3rd
Panel control 2nd 3rd
postback 2nd 3rd 4th
tables
DataGrid class 2nd 3rd
HtmlTable control 2nd
HtmlTableCell control 2nd
HtmlTableCellCollection control 2nd
HtmlTableRow control 2nd
HtmlTableRowCollection control 2nd
Table control 2nd
TableCell control 2nd 3rd
TableRow control 2nd 3rd
text
Label control 2nd 3rd
TextBox class 2nd
text. [See text]2nd [See text]
tracing
categorizing output 2nd 3rd 4th 5th 6th 7th
enabling tracing 2nd 3rd 4th 5th 6th 7th 8th
equipping pages for Trace mode 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
trace information categories 2nd
View State 2nd 3rd 4th
EnableViewState/EnableViewStateMac attributes (Page object) 2nd
Web controls. [See Web controls]
Web.Config/Machine.Config page settings 2nd 3rd
Panel class 2nd 3rd
parameters
passing parameters to stored procedures (ADO.NET) 2nd 3rd 4th 5th 6th
Passport authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th
Passport Web site

This document is created with the unregistered version of CHM2PDF Pilot

PassportIdentity class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
attributes 2nd
Site IDs
web.config setup 2nd
Passport User ID (PUID) 2nd 3rd 4th
PassportIdentity class 2nd
path attribute
forms authentication
paths
rewriting paths via HttpModule 2nd 3rd 4th 5th 6th
Perfmon tool. [See performance monitors]
performance monitors
creating custom monitors 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
creating monitor categories 2nd 3rd 4th 5th 6th
deleting monitor categories 2nd 3rd
sending application information to monitors via PerformanceCounter object 2nd 3rd 4th 5th
Windows Performance Monitor utility 2nd
listings
3.7, initializing new monitor category and monitor object 2nd
3.8, incrementing performance monitor using PerformanceCounter object 2nd 3rd 4th
3.9, deleting a performance category 2nd
PerformanceCounter class 2nd
persistence
server controls 2nd 3rd 4th 5th 6th 7th
PostalCode attribute (PassportIdentity class)
postback
encryption/SSL
IPostBackEventHandler
server controls 2nd 3rd 4th 5th
Web controls 2nd 3rd 4th
PreferredEmail attribute (PassportIdentity class)
PreRender event
Page object
Web controls
PreRender event (custom server controls)
presentation, separating from code using code behind 2nd 3rd 4th 5th 6th 7th 8th 9th
. [See also code behind]
listing 2.8, SamplePage3.aspx, using code behind 2nd 3rd
listing 2.9, SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd
principals 2nd
. [See also security]
GenericPrincipal class 2nd
GenericPrincipal object 2nd 3rd 4th 5th 6th 7th 8th 9th
IPrincipal interface 2nd
WindowsPrincipal class 2nd 3rd
WindowsPrincipal object 2nd 3rd 4th 5th 6th
processModel settings (Web.Config/Machine.Config files) 2nd 3rd
proerties
ValidationEventArgs class
ProfileVersion attribute (PassportIdentity class)
programming controls
HTML controls

This document is created with the unregistered version of CHM2PDF Pilot

HtmlAnchor 2nd 3rd 4th 5th
HtmlButton 2nd 3rd 4th 5th
HtmlForm 2nd 3rd 4th 5th 6th 7th 8th 9th
HtmlImage 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
HtmlInputButton 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputCheckbox 2nd
HtmlInputFile 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputHidden 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputImage 2nd
HtmlInputRadioButton 2nd
HtmlInputText 2nd
HtmlSelect 2nd
HtmlTable 2nd
HtmlTableCell 2nd
HtmlTableCellCollection 2nd
HtmlTableRow 2nd
HtmlTableRowCollection 2nd
HtmlTextArea 2nd
Web forms controls 2nd
properties
. [See also classes]
AdRotator class
Button class
Cache class
CacheDependency class
Calendar class
CheckBox class
Control class
DataAdapter class (ADO.NET)
DataColumn class (ADO.NET)
DataGrid class 2nd
DataList class 2nd
DataRelation class (ADO.NET)
DataSet class (ADO.NET)
DataTable class (ADO.NET)
Debug class
DropDownList class
EventLog class
HtmlAnchor control
HtmlButton control
HtmlForm control
HtmlImage control
HtmlInputButton control
HtmlInputCheckBox control
HtmlInputFile control
HtmlInputHidden control
HtmlInputImage control
HtmlInputRadioButton control
HtmlInputText control
HtmlSelect control
HtmlTable control
HtmlTableCell control

This document is created with the unregistered version of CHM2PDF Pilot

HtmlTableCellCollection control
HtmlTableRow control
HtmlTableRowCollection control
HtmlTextArea control
HttpApplication class
HttpCachePolicy class
HttpCacheVaryByHeaders class
HttpCacheVaryByParams class
HttpRequest class
HttpResponse class
HttpServerUtility class
HttpSessionState class 2nd
Hyperlink class 2nd
ImageButton class
Label class
LinkButton class
ListBox class
OleDbCommand class (ADO.NET)
PerformanceCounter class
RadioButton class
Repeater class
server controls
basic server control that stores property state 2nd
SqlConnectionclass (ADO.NET)
SqlDataAdapter class (ADO.NET)
SqlDataReader class (ADO.NET)
SqlError (ADO.NET)
SqlException (ADO.NET)
Table class
TableCell class
TableRow class
TextBox class
TraceContext class
user controls 2nd 3rd 4th
WebControl class
XmlDocument class
XmlNamedNodeMap class
XmlNode class
XmlNodeList class
XmlNodeReader class 2nd
XmlReader class
XmlTextReader class
XmlTextWriter class
XmlValidatingReader class
XmlWriter class
propeties
Page class
Panel class
protection attribute
forms authentication 2nd
proxies
Web services 2nd 3rd 4th 5th

This document is created with the unregistered version of CHM2PDF Pilot

creating with Visual Studio .NET Web References 2nd 3rd 4th 5th 6th 7th 8th
creating with wsdl.exe 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
SoapHttpClientProtocol 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
SoapHttpClientProtocol CookieContainer property 2nd 3rd 4th 5th 6th 7th 8th 9th
PUID (Passport User ID) 2nd 3rd 4th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Q switch (XCOPY)
queries
caching 2nd 3rd 4th 5th 6th 7th
querying databases 2nd 3rd 4th
(begin)
(end)
command objects 2nd
executing commands that donÕt return data (nonquery commands) 2nd 3rd 4th
executing text-based commands to retrieve data reader objects 2nd 3rd 4th 5th 6th
displaying query data in Web browsers 2nd 3rd 4th 5th
displaying query data in Web Forms DataGrid control 2nd
displaying query results using SqlDataAdapter and DataSet 2nd 3rd
executing stored procedures that return scalar values 2nd 3rd
passing parameters to stored procedures 2nd 3rd 4th 5th 6th
retrieving data reader objects via text commands 2nd 3rd 4th 5th 6th 7th
retrieving data via stored procedures 2nd 3rd 4th 5th
SqlDataReader object 2nd
returning via text commands 2nd 3rd 4th 5th 6th
querying XML data (XPath expressions) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th
common XPath scenarios
dot (.) operator
inequality operator (!=)
listings
10.24, XPath query page used to test XPath queries against books.xml 2nd 3rd
10.25, running a simple query 2nd
10.26, retrieving author information based on author ID 2nd
10.27, retrieving multiple instances of same author 2nd 3rd
10.28, combining multiple criteria with AND and OR 2nd
10.29, excluding nodes based on attribute values 2nd
10.30, retrieving a specific TITLE node by querying on its text 2nd
10.31, retrieving a specific BOOK node by querying on text in BookÕs TITLE node 2nd
manipulating current nodes using XPathNodeIterator Current property 2nd 3rd
retrievig the root node of a document
W3C XPath Web site
question mark (?)
authorization settings (Web.Config/Machine.Config files)
URL authorization

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

R switch (XCOPY)
radio buttons
HtmlInputRadioButton control 2nd
RadioButton class 2nd 3rd 4th
RaisePostBackEvent (custom server controls)
RaisePostDataChangedEvent (custom server controls)
raising events via HttpModules 2nd 3rd 4th 5th 6th
RangeValidator control 2nd 3rd 4th
reading XML data
XmlDataReader object 2nd 3rd 4th 5th 6th 7th
XmlNode object 2nd 3rd 4th 5th
XmlNodeReader object 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th 20th
changing values 2nd 3rd 4th 5th 6th 7th
navigating documents 2nd 3rd
XPath queries. [See XML, XPath queries]
XPath queries 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th
common XPath scenarios
dot (.) operator
inequality operator (!=)
listing 10.24, XPath query page used to test XPath queries against books.xml 2nd 3rd
listing 10.25, running a simple query 2nd
listing 10.26, retrieving author information based on author ID 2nd
listing 10.27, retrieving multiple instances of same author 2nd 3rd
listing 10.28, combining multiple criteria with AND and OR 2nd
listing 10.29, excluding nodes based on attribute values 2nd
listing 10.30, retrieving a specific TITLE node by querying on its text 2nd
listing 10.31, retrieving a specific BOOK node by querying on text in BookÕs TITLE node 2nd
manipulating current nodes using XPathNodeIterator Current property 2nd 3rd
retrievig the root node of a document
W3C XPath Web site
refreshing
cached items via dependencies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
Region attribute (PassportIdentity class)
Register directive
server controls 2nd
user controls 2nd 3rd
registering controls
@Register directive (Page object)
RegularExpressionValidator control 2nd 3rd
RelAbsolute.aspx code listing 2nd 3rd
remote
tracing
viewing application trace data from remote browser windows 2nd 3rd
Repeater class 2nd 3rd 4th 5th 6th 7th
Request object
HttpBrowserCapabilities class 2nd 3rd 4th 5th 6th 7th
HttpRequest class 2nd
RequiredFieldValidation control 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

RequiredFieldValidator control 2nd 3rd 4th
reusable code 2nd 3rd
custom user-interface objects. [See controls]
server-side includes (SSIs) 2nd
reusing code. [See components]
rewriting paths via HttpModule 2nd 3rd 4th 5th 6th
rich clients 2nd
roles
forms authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
UserRoleMappings table relationships
Roles tables
rotating advertisements (AdRotator class) 2nd 3rd 4th 5th 6th
banner dimensions
listing 2.29 2nd 3rd 4th
listing 2.30 2nd
rows
HtmlTableRow control 2nd
HtmlTableRowCollection control 2nd
HtmlTextArea control 2nd
rows. [See tables]
running database queries 2nd 3rd 4th
(begin)
(end)
command objects 2nd
executing commands that donÕt return data (nonquery commands) 2nd 3rd 4th
executing text-based commands to retrieve data reader objects 2nd 3rd 4th 5th 6th
displaying query data in Web browsers 2nd 3rd 4th 5th
displaying query data in Web Forms DataGrid control 2nd
displaying query results using SqlDataAdapter and DataSet 2nd 3rd
executing stored procedures that return scalar values 2nd 3rd
passing parameters to stored procedures 2nd 3rd 4th 5th 6th
retrieving data reader objects via text commands 2nd 3rd 4th 5th 6th 7th
retrieving data via stored procedures 2nd 3rd 4th 5th
SqlDataReader object 2nd
returning via text commands 2nd 3rd 4th 5th 6th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

S switch (XCOPY) 2nd
SalesByCategory stored procedure (Northwind Database)
passing parameters to stored procedures 2nd 3rd 4th 5th
SalesByEmployee.aspx code listing 2nd 3rd
SalesPeople.ascx code listing 2nd 3rd
SamplePage2.aspx listing, event handler for a button 2nd
SamplePage3.aspx listing, using code behind 2nd 3rd
SAX (Simple API for XML) 2nd 3rd
scalar vaules
executing stored procedures that return scalar values (ADO.NET) 2nd 3rd
scavenging
schemas
SQL Server state
schemas (XML) 2nd
case sensitivity
complex types 2nd 3rd 4th 5th 6th 7th
creating
W3C XML schemas 2nd 3rd 4th 5th 6th 7th 8th
XSD schemas 2nd 3rd
Document Type Definitions (DTDs) 2nd 3rd 4th
listings
10.34, W3C schema definition boilerplate 2nd
10.35, W3C schema containing a definition for a simple type 2nd
10.36, W3C schema containing an element definition that refers to a type definition 2nd
10.37, W3C schema containing a complex type that refers to a simple type 2nd
simple types 2nd 3rd 4th 5th 6th 7th
validating documents using W3C schemas 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
XML Data-Reduced (XDR) schemas 2nd 3rd 4th 5th
Search dialog box server control
creating composite Search server control by overriding the CreateChildControls method of the Control 2nd 3rd
page created to host an instance of composite Search server control 2nd
using delegation to provide access to Text property of child control 2nd 3rd
Search dialog box user control 2nd 3rd 4th
. [See also user controls]
customizing 2nd 3rd 4th
handling events 2nd 3rd 4th
Register directive 2nd 3rd 4th 5th
TitleBarText property 2nd 3rd 4th
Search dialog box user controlShowAdvanced method 2nd 3rd 4th
section handlers (Web.Config/Machine.Config) 2nd
authentication 2nd 3rd 4th
authorization 2nd
browser capabilities 2nd
compilation 2nd 3rd
custom errors 2nd 3rd
execution timeout 2nd
globalization 2nd 3rd
HTTP handlers
HTTP modules 2nd

This document is created with the unregistered version of CHM2PDF Pilot

pages 2nd 3rd
process model 2nd 3rd
session state 2nd 3rd
trace 2nd 3rd 4th
Web services 2nd 3rd
Secure Socket Layer (SSL)
postback information
security 2nd
authentication
cookie-based authentication
HttpModules 2nd 3rd 4th
Passport authentication 2nd 3rd
Web.Config/Machine.Config settings 2nd 3rd
Windows authentication
authorization
Web.Config/Machine.Config settings 2nd
file authorization 2nd
forms authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
advantages 2nd
process flow 2nd 3rd 4th 5th
roles 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
settings 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
versus Windows authentication 2nd 3rd
identities 2nd
GenericIdentity class 2nd 3rd
PassportIdentity class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
WindowsIdentity class 2nd
impersonation 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
determining identity with WindowsIdentity 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
hard-coded impersonation 2nd 3rd 4th 5th 6th 7th
ISAPI filter
listings
7.1, user authentication with WindowsPrincipal 2nd 3rd 4th
7.10, web.config file that uses URL authorization Location element 2nd 3rd 4th
7.11, Web.config that maps roles to application locations 2nd 3rd
7.12, login page that authenticates only two users 2nd 3rd 4th
7.13, login page that authenticates only two users, class file 2nd 3rd 4th
7.14, global.asax class that handles the AuthorizeRequest application event 2nd 3rd
7.15, web.config file setting authentication to forms and defining URL authorization 2nd
7.16, HTML for login.aspx 2nd 3rd
7.17, HTML for login.aspx, class file 2nd 3rd 4th 5th
7.18, Transact SQL to create tables and stored procedures used by login.aspx 2nd 3rd 4th 5th
7.19, global.asax containing the Application_AuthenticateRequest handler 2nd 3rd
7.2, user authentication with WindowsPrincipal, class file 2nd
7.20, Webform1.aspx 2nd 3rd 4th 5th
7.21, Webform1.aspx, code behind class 2nd 3rd 4th 5th
7.22, Web.config set up for impersonation 2nd 3rd
7.23, Web.config with a hard-coded impersonation account 2nd 3rd
7.3, Web.config for simple forms authentication 2nd 3rd
7.4, simple login page 2nd 3rd 4th 5th
7.5, simple login page, class file 2nd 3rd 4th 5th 6th
7.6, security;web.config setup for Passport authentication 2nd

This document is created with the unregistered version of CHM2PDF Pilot

7.7, Web User Control That Wraps the Display of the Passport Logo 2nd 3rd 4th
7.8, login.aspx page shown when users attempt to access content when they are not authenticated 2nd 3rd
7.9, page that displays the PUID, e-mail address, and member name of a logged-in Passport user 2nd 3rd
locking down Web configuration settings 2nd
Passport authentication 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
PassportIdentity class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
Site IDs
web.config setup 2nd
principals 2nd
GenericPrincipal class 2nd
GenericPrincipal object 2nd 3rd 4th 5th 6th 7th 8th 9th
IPrincipal interface 2nd
WindowsPrincipal class 2nd 3rd
WindowsPrincipal object 2nd 3rd 4th 5th 6th
URL authorization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
location element 2nd 3rd 4th 5th
using forms authentication with URL authorization 2nd 3rd 4th
FormsAuthenticationTicket class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
validation
validating documents using W3C schemas 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
Windows authentication 2nd
versus forms authentication 2nd 3rd
WindowsBuiltInRole enumeration 2nd 3rd 4th 5th 6th 7th
selecting
HtmlSelect control 2nd
separating code from presentation using code behind 2nd 3rd 4th 5th 6th 7th 8th 9th
. [See also code behind]
listing 2.8, SamplePage3.aspx, using code behind 2nd 3rd
listing 2.9, SimplePage3.aspx.vb, code behind file for SimplePage3.aspx 2nd
serialization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th
(Web services)
DataSet 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
WebMethod returning a DataTable 2nd 3rd 4th 5th
WebMethod returning a Master-Detail relationship 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
server controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
binding controls to data 2nd
Calendar class 2nd 3rd 4th 5th
CheckBox class 2nd 3rd 4th
compiling controls as .NET components 2nd 3rd 4th 5th 6th
creating components with command-line compiler 2nd 3rd 4th 5th 6th
deploying components 2nd
Control class 2nd 3rd
creating basic server controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
creating composite controls 2nd 3rd 4th 5th 6th 7th 8th
DataGrid class 2nd 3rd
DataList class 2nd 3rd 4th 5th 6th
listing 2.33, DataList server control bound to a Hashtable object 2nd 3rd
DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
events 2nd 3rd 4th
generating postback 2nd 3rd 4th 5th
HTML controls 2nd
. [See also HTML controls]

This document is created with the unregistered version of CHM2PDF Pilot

Hyperlink class 2nd 3rd
Image class 2nd 3rd
ImageButton class 2nd 3rd 4th
LinkButton class 2nd 3rd
ListBox class 2nd 3rd
listings
9.10, batch file that compiles and deploys a .NET component project automatically 2nd
9.11, composite control 2nd 3rd
9.12, page created to host an instance of composite Search server control 2nd
9.13, using delegation to provide access to Text property of child control 2nd 3rd
9.14, CustomTextBox control that contains set of custom formatting defaults 2nd
9.15, ASP.NET Page that uses an instance of subclassed CustomText control 2nd 3rd
9.16, hyperlink control that can generate client-side postback of a form 2nd 3rd
9.17, basic control that stores property state 2nd
9.18, page to contain counter control 2nd 3rd
9.19, rewritten CurrentValue property, demonstrating lack of state persistence 2nd
9.7, Hello World server control example 2nd 3rd
9.8, ASP.NET page that references custom Hello World server control 2nd 3rd
9.9, namespace and class that can be compiled into a .NET component 2nd
Panel class 2nd 3rd
persistence support 2nd 3rd 4th 5th 6th 7th
RadioButton class 2nd 3rd 4th
Register directive 2nd
Repeater class 2nd 3rd 4th 5th 6th 7th
rich clients 2nd 3rd 4th
subclassing existing controls 2nd 3rd 4th 5th
supporting designers 2nd
Table class 2nd
TableCell class 2nd 3rd
TableRow class 2nd 3rd
TextBox class 2nd
validation controls 2nd 3rd 4th 5th 6th
CompareValidator 2nd 3rd 4th 5th
CustomValidator 2nd 3rd 4th 5th
multiple validators
RangeValidator 2nd 3rd 4th
RegularExpressionValidator 2nd 3rd
RequiredFieldValidator 2nd 3rd 4th
summarizing validation errors 2nd 3rd 4th
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
server-side control 2nd
server-side cursors 2nd
server-side includes (SSIs)
disadvantages 2nd
Global.asax files 2nd
server-side object tags
Global.asax files 2nd 3rd 4th 5th
servers
HttpServerUtility class 2nd
services. [See Web services]
Session object
Global.asax event handler 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

session state
cookieless session management
rewriting paths via HttpModule 2nd 3rd 4th 5th 6th
EnableSessionState attribute (Page object)
HttpSessionState class 2nd 3rd
Web services 2nd 3rd 4th
Web.Config/Machine.Config sessionState settings 2nd 3rd
sessions 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
. [See also state management]
choosing between session states 2nd 3rd 4th 5th 6th
HttpSessionState class 2nd
identity
cookie-based 2nd 3rd
cookieless 2nd 3rd 4th 5th 6th
in process session state 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
out of process session state (State Server) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
storing state information in SQL Server 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
using session state 2nd 3rd
adding values 2nd 3rd 4th
cleaning up after session state 2nd
initializing user states 2nd
retrieving values 2nd
setting values 2nd 3rd
Web farms 2nd 3rd 4th 5th 6th 7th 8th
Simple API for XML (SAX) 2nd 3rd
Simple Object Access Protocol (SOAP) 2nd 3rd
simple types (XML) 2nd 3rd 4th 5th 6th 7th
SimpleDataSet
binding to forms 2nd 3rd 4th 5th
running wsdl.exe 2nd 3rd 4th
Web Form client 2nd 3rd 4th
SimpleDataSet (Web services) 2nd 3rd 4th 5th 6th
SimplePage.asp listing 2nd 3rd
SimplePage.aspx listing, a reworking of Listing 2.1 in ASP.NET 2nd 3rd
SimplePage2.asp listing, showing code to preserve form state in ASP.OLD 2nd 3rd
SimplePage3.asp listing, reaction to user interaction in ASP.old 2nd 3rd
SimplePage3.aspx.vb listing, code behind file for SimplePage3.aspx 2nd 3rd
single-threaded apartment (STA) threading 2nd
slash (/)
path attribute, forms authentication 2nd
slash (/) character
XML closing tags
SlidingExpiration argument (Cache class) 2nd 3rd 4th 5th
SmartNavigation attribute, Page object
SmtpMailclass 2nd 3rd
SOAP (Simple Object Access Protocol) 2nd 3rd
SoapHttpClientProtocol class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
CookieContainer property 2nd 3rd 4th 5th 6th 7th 8th 9th
Split( ) function (forms authentication)
SQL
combining with XML (HttpHandlers) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
SqlHandler 2nd 3rd 4th 5th 6th 7th

This document is created with the unregistered version of CHM2PDF Pilot

SqlHandler output
XSL template used to format output 2nd 3rd 4th 5th
SQL Server
connecting to databases 2nd
storing state information 2nd 3rd 4th 5th
advantages/disadvantages 2nd 3rd 4th 5th 6th 7th 8th 9th
tempdb
SqlCommand class
SqlCommand object
calling stored procedures 2nd 3rd 4th
SQLCommand object
ExecuteScalar method 2nd 3rd
SqlCommand object
Parameters collection 2nd
SqlConnection class 2nd 3rd 4th 5th 6th 7th
SqlConnection class (ADO.NET) 2nd
SqlConnection object 2nd 3rd 4th
SqlDataAdapter
. [See also data adapters (ADO.NET)]
SqlDataAdapter class (ADO.NET) 2nd
SqlDataAdapter object
. [See also data adapters (ADO.NET)]
SqlDataReader class (ADO.NET) 2nd 3rd
SqlDataReader object 2nd 3rd 4th 5th 6th
calling stored procedures 2nd 3rd 4th
returning via text commands 2nd 3rd 4th 5th 6th 7th
SqlError class (ADO.NET) 2nd
SqlException class (ADO.NET) 2nd
Src attribute, Page object 2nd
SSIs (server-side includes)
disadvantages 2nd
Global.asax files 2nd
SSL (Secure Socket Layer)
postback information
STA (single-threaded apartment) threading 2nd
standards
Web services 2nd
state
form state 2nd 3rd
persistence
server controls 2nd 3rd 4th 5th 6th
session state
EnableSessionState attribute (Page object)
HttpSessionState class 2nd 3rd
Web.Config/Machine.Config sessionState settings 2nd 3rd
View State
EnableViewState attribute (Page object) 2nd
EnableViewStateMac attribute (Page object) 2nd
Web controls 2nd 3rd 4th
Web service-based state service 2nd 3rd 4th
state management 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
ASP 1.0 2nd

This document is created with the unregistered version of CHM2PDF Pilot

browser requests 2nd 3rd 4th 5th 6th
Choosing Between Session States 2nd 3rd 4th 5th 6th
cookies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
cookie-based session identity 2nd 3rd
cookieless session identity 2nd 3rd 4th 5th 6th
in process session state 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
listings
5.1, HTTP request using Telnet 2nd 3rd
5.10, timing read/write operations on session state 2nd 3rd 4th
5.2, Set-Cookie header 2nd
5.3, relative and absolute references 2nd 3rd
5.4, setting a session value 2nd
5.5, getting a session value 2nd
5.6, Session Start event 2nd
5.7, adding/displaying session values 2nd 3rd
5.8, out of process session state 2nd
5.9, starting State Server 2nd
out of process session state (State Server) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
storing state information in SQL Server 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
using session state 2nd 3rd
adding values 2nd 3rd 4th
cleaning up after session state 2nd
initializing user states 2nd
retrieving values 2nd
setting values 2nd 3rd
Web farms 2nd 3rd 4th 5th 6th 7th 8th
State Server 2nd 3rd 4th 5th 6th 7th 8th 9th
advantages/disadvantages 2nd 3rd 4th 5th
StaticFileHandler HttpHandler
. [See also HttpHandlers]
stored procedures
executing stored procedures that return data 2nd 3rd 4th 5th
passing parameters 2nd 3rd 4th 5th 6th
returning scalar values 2nd 3rd
versus text-based commands
style sheets
transforming XML documents 2nd 3rd 4th 5th 6th 7th
associating documents with style sheets 2nd 3rd
creating style sheets 2nd
creating XSLT style sheets 2nd 3rd 4th
editing XSLT files with Visual Studio .NET
programmatically performing XSL transformations 2nd 3rd
subclassing existing server controls 2nd 3rd 4th 5th
subdirectories
. [See also directories]
summarizing validation errors 2nd 3rd 4th
switches
XCOPY deployment 2nd 3rd 4th 5th
/-Y
/A
/C
/D 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

/E 2nd
/EXCLUDE 2nd
/EXCLUDE;
/F
/H
/I
/K
/L
/M
/N
/O
/P
/Q
/R
/S 2nd
/T
/U
/V
/W
/X
/Y 2nd 3rd
/Z
System.Diagnostics
Debug class 2nd 3rd
EventLog class 2nd 3rd 4th 5th 6th 7th 8th 9th
PerformanceCounter 2nd
System.Web
TraceContext 2nd
System.Web section (Web.Config files)
authentication settings 2nd 3rd 4th
authorization settings 2nd
browser capabilities settings 2nd
compilation settings 2nd 3rd
custom error settings 2nd 3rd
execution timeout settings 2nd
globalization settings 2nd 3rd
HTTP handler settings
HTTP module settings 2nd
page settings 2nd 3rd
process model settings 2nd 3rd
session state settings 2nd 3rd
Trace settings 2nd 3rd 4th
Web services settings 2nd 3rd
System.Web.UI.Control
events 2nd

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

T switch (XCOPY)
Table class 2nd
TableCell class 2nd 3rd
TableRow class 2nd 3rd
tables
DataGrid class 2nd 3rd
hash tables. [See hash tables]
HtmlTable control 2nd
HtmlTableCell control 2nd
HtmlTableCellCollection control 2nd
HtmlTableRow control 2nd
HtmlTableRowCollection control 2nd
HtmlTextArea control 2nd
Table control 2nd
TableCell control 2nd 3rd
TableRow control 2nd 3rd
tags
XML. [See elements (XML)]
Telnet
HTTP requests 2nd 3rd 4th 5th
tempdb (SQL Server)
text
hiding
HtmlInputHidden control 2nd 3rd 4th 5th 6th 7th 8th
HtmlInputText control 2nd
HtmlTextArea control 2nd
Label control 2nd 3rd
TextBox class 2nd
XmlTextReader class 2nd 3rd
XmlTextWriter class 2nd
writing data 2nd 3rd 4th 5th 6th 7th
text-based commands
retrieving data reader objects 2nd 3rd 4th 5th 6th 7th
versus stored procedures
TextBox class 2nd
TextPad text editor
tilde (~) character
server side control HREFs
time stamping output HTML (BeginRequest-EndRequest module) 2nd 3rd 4th 5th
timeout attribute
forms authentication 2nd 3rd 4th
TimeOut property (HttpSessionState class) 2nd
Timing.aspx code listing 2nd 3rd
Trace attribute, Page object
TraceContext class 2nd
TraceHandler HttpHandler
. [See also HttpHandlers]
TraceMode attribute, Page object
tracing 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th

This document is created with the unregistered version of CHM2PDF Pilot

activating tracing via Trace attribute (Page object)
enabling application tracing 2nd 3rd 4th
enabling page tracing 2nd 3rd 4th
equipping pages for Trace mode 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
assigning output categories 2nd 3rd 4th
listings
3.1, simple page equipped for tracing with calls to Trace.Write 2nd 3rd 4th
3.2, categorized Trace.Write output 2nd 3rd
3.3, using Web.config to activate tracing for entire Web directories 2nd
sorting trace entries via TraceMode attribute (Page object)
trace information categories 2nd
viewing application trace data from remote browser windows 2nd 3rd
Web.Config/Machine.Config Trace settings 2nd 3rd 4th
Transaction attribute, Page object
TransactionOption property (WebMethodAttribute class) 2nd
transforming XML documents using style sheets 2nd 3rd 4th 5th 6th 7th
associating documents with style sheets 2nd 3rd
creating style sheets 2nd
creating XSLT style sheets 2nd 3rd 4th
editing XSLT files with Visual Studio .NET
programmatically performing XSL transformations 2nd 3rd

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

U switch (XCOPY)
UDDI (Universal Description, Discovery, and Integration) 2nd
UICulture attribute, Page object
uninstalling components from GAC 2nd
Universal Description, Discovery, and Integration (UDDI) 2nd
Unload event
  Page object
  Web controls
Unload event (custom server controls)
updating XML documents using XmlNodeReader class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
  changing values 2nd 3rd 4th 5th 6th 7th
  navigating documents 2nd 3rd
  XPath queries. [See XML, XPath queries]
uplevel/downlevel rendering 2nd
uploading files
  HtmlInputFile control 2nd 3rd 4th 5th 6th 7th 8th
URL authorization 2nd 3rd 4th 5th
URL authorization
URL authorization 2nd 3rd 4th
URL authorization
  location element 2nd 3rd 4th 5th
  using forms authentication with URL authorization 2nd 3rd 4th
    FormsAuthenticationTicket class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
URLs
  location representation 2nd 3rd 4th
  loginurl attribute
    forms authentication
  rewriting paths via HttpModule 2nd 3rd 4th 5th 6th
URLs (uniform resource locators) 2nd 3rd 4th
user controls 2nd 3rd 4th 5th 6th 7th 8th 9th
  adding methods 2nd 3rd 4th 5th
  adding properties 2nd 3rd 4th
  Control directive 2nd
  handling events 2nd 3rd
  listings
    9.1, Search dialog box 2nd 3rd
    9.2, Search dialog box with TitleBarText property 2nd 3rd
    9.3, Register directive referencing a user control 2nd
    9.4, custom control utilized in an ASP.NET Web Form 2nd
    9.5, Search control with ShowAdvanced method 2nd 3rd
    9.6, handling events 2nd 3rd
  programming controls in pages 2nd 3rd 4th
  Register directive 2nd 3rd
user intercases
  Web controls. [See Web controls]
User tables
UserData member (FormsAuthenticationTicket class) 2nd 3rd
UserRoleMappings tables
users

This document is created with the unregistered version of CHM2PDF Pilot

initializing user states 2nd
. [See also state management]

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

V switch (XCOPY)
validation
validating documents using W3C schemas 2nd 3rd 4th 5th 6th
validating XML documents using W3C schemas 2nd 3rd 4th 5th 6th 7th 8th
XmlValidatingReader class 2nd 3rd
validating schemas 2nd 3rd 4th 5th 6th 7th 8th
validation controls (validators) 2nd 3rd 4th 5th 6th
CompareValidator 2nd 3rd 4th 5th
properties
ValidationCompareOperator enumeration 2nd 3rd
CustomValidator 2nd 3rd 4th 5th
multiple validators
RangeValidator 2nd 3rd 4th
RegularExpressionValidator 2nd 3rd
RequiredFieldValidator 2nd 3rd 4th
summarizing validation errors 2nd 3rd 4th
ValidationCompareOperator enumeration 2nd 3rd
ValidationCompareOperator operators 2nd 3rd
ValidationEventArgs class 2nd 3rd
VaryBrowserBroken.aspx code listing
VaryBrowserFixed.aspx code listing 2nd
VaryByCustom attribute (OutputCache directive) 2nd 3rd 4th 5th
VaryByCustom caching 2nd 3rd
VaryByHeader attribute (OutputCache directive) 2nd 3rd
VaryByHeader caching 2nd 3rd
VaryByParam attribute (OutputCache directive) 2nd 3rd 4th 5th
VaryByParam caching 2nd 3rd 4th 5th 6th 7th
VaryByParams attribute (OutputCache directive) 2nd
View State
EnableViewState attribute (Page object) 2nd
EnableViewStateMac attribute (Page object) 2nd
Web controls 2nd 3rd 4th
Visual Studio
creating .NET components
Visual Studio .NET
creating XSD schemas 2nd 3rd
editing XSLT files
Web References 2nd 3rd 4th 5th 6th 7th 8th
Visual Studio.NET
editing Web configuration files 2nd
Web Server Controls
. [See also server controls]

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

W switch (XCOPY)
W#C (World Wide Web Consortium)
  XPath Web site
W3C
  Web pages
    SOAP (Simple Object Access Protocol) 2nd
  XML DTD Web site
  XML schemas Web site
W3C XML schemas
  case sensitivity
  creating 2nd 3rd 4th 5th 6th 7th 8th
  listings
    10.34, W3C schema definition boilerplate 2nd
    10.35, W3C schema containing a definition for a simple type 2nd
    10.36, W3C schema containing an element definition that refers to a type definition 2nd
    10.37, W3C schema containing a complex type that refers to a simple type 2nd
  validating documents 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
W3C XSL Web site
W3C XSLT Web site
Wallet attribute (PassportIdentity class)
WarningLevel attribute, Page object 2nd
Web browsers
  page requests 2nd 3rd 4th 5th 6th
Web controls 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th 20th 21st 22nd 23rd 24th 25th 26th
  AdRotator class 2nd 3rd 4th 5th 6th
    AdvertisementFile property 2nd
    banner dimensions
    listing 2.29 2nd 3rd
    listing 2.30 2nd
  Button class 2nd 3rd 4th
  Calendar class 2nd 3rd 4th 5th
  Calendar.aspx 2nd 3rd 4th 5th 6th 7th 8th
    output
    rendered HTML source 2nd 3rd 4th 5th 6th 7th
  CheckBox class 2nd 3rd 4th
  data binding
    . [See also data binding]
  DataGrid class 2nd 3rd
  DataList class 2nd 3rd 4th 5th 6th
    listing 2.33, DataList server control bound to a Hashtable object 2nd 3rd
  determining browser capabilities 2nd 3rd 4th 5th 6th 7th
  DropDownList class 2nd 3rd 4th 5th 6th 7th 8th
  equivalent HTML tags 2nd 3rd
  event model 2nd 3rd
  HttpApplication class 2nd 3rd
  HttpBrowserCapabilities class 2nd 3rd 4th 5th 6th 7th
  HttpRequest class 2nd
  HttpResponse class 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

HttpServerUtility class 2nd
HttpSessionState class 2nd 3rd
Hyperlink class 2nd 3rd
Image class 2nd 3rd
ImageButton class 2nd 3rd 4th
Label class 2nd 3rd
LinkButton class 2nd 3rd
ListBox class 2nd 3rd
mobile controls 2nd
Panel class 2nd 3rd
postback 2nd 3rd 4th
programming (creating an ASP.NET Web form) 2nd
RadioButton class 2nd 3rd 4th
Repeater class 2nd 3rd 4th 5th 6th 7th
Table class 2nd
TableCell class 2nd 3rd
TableRow class 2nd 3rd
TextBox class 2nd
View State 2nd 3rd 4th
WebControl class 2nd 3rd
Web farms 2nd
state management 2nd 3rd 4th 5th 6th 7th 8th
. [See also state management]
Web Forms
binding objects via data adapters 2nd 3rd 4th
DataView object 2nd 3rd 4th
calling NorthwindOrders Web service asynchronously 2nd 3rd 4th 5th
Web forms
calling Web services 2nd 3rd 4th 5th 6th 7th 8th 9th
Web Forms
data binding
. [See also data binding]
data entry 2nd 3rd 4th
comparison validation 2nd 3rd 4th 5th
custom validation 2nd 3rd 4th 5th
range validation 2nd 3rd 4th
regular expression validation 2nd 3rd
required field validation 2nd 3rd 4th
summarizing validation errors 2nd 3rd 4th
using multiple validators
validation 2nd 3rd 4th
Web forms
grouping controls (Panel class) 2nd 3rd
Web Forms
listings
2.25, Web form declaration
mobile controls 2nd
postback 2nd 3rd 4th
user controls 2nd 3rd 4th 5th 6th 7th
adding methods 2nd 3rd 4th 5th
adding properties 2nd 3rd 4th
handling events 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

programming controls in pages 2nd 3rd 4th
View State 2nd 3rd 4th
Web forms controls. [See Web controls]
Web pages
caching 2nd 3rd 4th 5th 6th 7th
. [See also caching]
browser differences 2nd 3rd 4th
partial pages 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
refreshing cached items via dependencies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
VaryByCustom/VaryByHeader 2nd 3rd
data representation
advertisement 2nd
HTML 2nd 3rd 4th 5th 6th 7th 8th 9th
location information 2nd 3rd
XML 2nd 3rd 4th 5th 6th 7th
headers
HttpCacheVaryByHeaders class 2nd
Set-Cookie 2nd
VaryByHeader caching 2nd 3rd
Web References (Visual Studio .NET) 2nd 3rd 4th 5th 6th 7th 8th
Web Service Description Language Tool (wsdl.exe) 2nd 3rd 4th
Web Service Description Language Tool (wsdl.exe), creating Web service proxies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
Web services 2nd 3rd 4th 5th 6th 7th
advantages 2nd
ASP.NET Web services 2nd 3rd 4th 5th 6th
calling Web method via HTTP Get 2nd
documentation pages 2nd 3rd
Hello World 2nd 3rd
serialization 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th
viewing WSDL output
WebMethod( ) attribute 2nd 3rd 4th
WebMethodAttribute class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
calling from Web forms 2nd 3rd 4th 5th 6th 7th 8th 9th
calling from Windows forms 2nd 3rd 4th 5th 6th 7th
consuming 2nd 3rd 4th 5th 6th 7th 8th 9th
asynchronous clients 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th
cookies 2nd 3rd 4th 5th 6th 7th 8th 9th
SoapHttpClientProtocol class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
data representation
advertisement 2nd
HTML 2nd 3rd 4th 5th 6th 7th
location information 2nd 3rd
XML 2nd 3rd 4th 5th
DataSets 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
WebMethod returning a DataTable 2nd 3rd 4th 5th
WebMethod returning a Master-Detail relationship 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
listings
6.1, sample invoice in HTML 2nd 3rd 4th 5th 6th
6.10, Web Method That Returns a Dataset with a Master Detail Relationship 2nd 3rd
6.11, XML Output from Calling the GetOrders WebMethod with a Date of 7/8/1996 2nd 3rd 4th 5th 6th 7th

This document is created with the unregistered version of CHM2PDF Pilot

6.12, Proxy Class (datasetsample.vb) for SimpleDataSet Generated with the WSDL Tool 2nd 3rd
6.13, Form That Is Bound to the SimpleDataSet Web Service 2nd 3rd 4th 5th
6.14, Proxy Created for SimpleDataSet by Visual Studio .NET 2nd 3rd 4th
6.15, Web Form Client for SimpleDataSet 2nd
6.16, Web Form That Calls the NorthwindOrder Web Service Asynchronously 2nd 3rd 4th 5th
6.17, Web Form That Calls the NorthwindOrders Service Asynchronously and Loads the Orders into a Gri
2nd 3rd 4th
6.18, Windows Form That Creates a Cookie Container and Utilizes the State Web Service 2nd 3rd 4th 5th
6th
6.2, sample invoice in XML 2nd 3rd
6.3, HelloWorld 2nd
6.4, Activator .asmx file 2nd
6.5, code-behind file for simple state service 2nd
6.6, Description property 2nd
6.7, Code Behind Web Service Class That Implements a Cached Time Service 2nd
6.8, service that returns ArrayList 2nd 3rd
6.9, Simple() WebMethod of SimpleDataSet 2nd 3rd 4th
proxies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
. [See also Web services, consuming]
creating with Visual Studio .NET Web References 2nd 3rd 4th 5th 6th 7th 8th
creating with wsdl.exe 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
session state 2nd 3rd 4th
standards 2nd
Web.Config/Machine.Config webServices settings 2nd 3rd
WebService class 2nd 3rd
Web Services Description Language (WSDL) 2nd
Web site URLs
Microsoft
XDR schemas
XML
Microsoft Knowledge Base
Windows 2000 command changes
Opera
Passport authentication
TextPad text editor
W3C
RFC 2109 - cookies
SOAP (Simple Object Access Protocol) 2nd
W3C XML DTD
W3C XML schemas
W3C XPath Web site
W3C XSL site
W3C XSLT site
Web.config
application-level debugging 2nd
hard-coded impersonation accounts 2nd 3rd
web.config
impersonation 2nd 3rd
loading BeginEnd HttpModule 2nd
Web.config
mapping roles to application locations 2nd 3rd
Web.Config

This document is created with the unregistered version of CHM2PDF Pilot

out of process session state 2nd
web.config
URL authorization Location element 2nd 3rd 4th
Web.config file
forms authentication 2nd 3rd
Web.config file setting authentication to forms and defining URL authorization 2nd
Web.Config files 2nd
accessing configuration files programmatically 2nd 3rd 4th
editing Web configuration files in Visual Studio.NET 2nd
listings
6. should be 5.
6.1, Location section to handle subdirectory settings 2nd 3rd
location section (configuring multiple locations) 2nd 3rd 4th 5th 6th
locking down settings 2nd
section handlers 2nd
authentication 2nd 3rd 4th
authorization 2nd
browser capabilities 2nd
compilation 2nd 3rd
custom errors 2nd 3rd
execution timeout 2nd
globalization 2nd 3rd
HTTP handlers
HTTP modules 2nd
pages 2nd 3rd
process model 2nd 3rd
session state 2nd 3rd
trace 2nd 3rd 4th
Web services 2nd 3rd
web.config Setup for Passport Authentication 2nd
WebControl class 2nd 3rd
WebMethod( ) attribute 2nd 3rd 4th
calling via HTTP GET
Simple() WebMethod of SimpleDataSet 2nd 3rd 4th 5th
viewing WSDL output
WebMethodAttribute class 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th
BufferResponse property 2nd
CacheDuration property 2nd 3rd 4th
Description property 2nd 3rd 4th 5th 6th 7th
EnableSession property 2nd 3rd 4th 5th
TransactionOption property 2nd
WebServiceHandlerFactory HttpHandler
. [See also HttpHandlers]
wildcards
*
HttpHandlers
Windows
rich clients 2nd
Windows 2000
state management 2nd
. [See also state management]
Windows authentication 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

versus forms authentication 2nd 3rd
WindowsBuiltInRole enumeration 2nd 3rd 4th 5th 6th 7th
Windows event log 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
Windows forms
calling Web services 2nd 3rd 4th 5th 6th 7th
Windows Forms
Web services
creating cookie containers/utilizing State Web service 2nd 3rd 4th 5th 6th 7th 8th
Windows Performance Monitor utility 2nd
.
creating custom monitors [See also performance monitors]
WindowsBuiltInRole enumeration 2nd 3rd 4th 5th 6th 7th 8th 9th
WindowsIdentity class 2nd
WindowsPrincipal class 2nd 3rd
WindowsPrincipal object 2nd 3rd 4th 5th 6th
WorkingWithCache.aspx code listing 2nd 3rd 4th 5th
World Wide Web Consortium (W3C)
XPath Web site
writing XML data
XmlTextWriter object 2nd 3rd 4th 5th 6th 7th
WSDL (Web Services Description Language) 2nd
WSDL output, Web services
wsdl.exe (Web Service Description Language Tool) 2nd 3rd 4th
wsdl.exe (Web Service Description Language Tool), creating Web service proxies 2nd 3rd 4th 5th 6th 7th 8th 9th 10th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

X switch (XCOPY)
XCOPY
listings
6.5, sample exclude file 2nd
using XCOPY for deployment 2nd 3rd
(begin)
(end)
(XCOPY deployment)
confirmations/overwriting files 2nd 3rd
deploying directory trees 2nd 3rd 4th 5th
deploying only modified files 2nd 3rd
deploying single directories 2nd
excluding files 2nd 3rd 4th
switches 2nd 3rd 4th 5th 6th 7th 8th
XDR (XML Data-Reduced) schemas 2nd 3rd 4th 5th
XML 2nd 3rd 4th 5th 6th
(begin)
(end)
(eXtensible Markup Language)
ADO.NET support 2nd
AdRotator control
advantages 2nd 3rd
AppendChild method 2nd
attributes
displaying attribute values using GetAttribute method 2nd 3rd
using attributes in queries 2nd 3rd 4th
classes
ChildNodes 2nd 3rd 4th 5th
ValidationEventArgs 2nd 3rd
XmlDocument 2nd 3rd 4th 5th 6th 7th 8th
XmlNamedNodeMap 2nd
XmlNode 2nd 3rd 4th
XmlNode, InsertAfter method 2nd 3rd
XmlNode, viewing document data 2nd 3rd 4th 5th 6th 7th 8th
XmlNodeList 2nd
XmlNodeReader 2nd 3rd 4th
XmlNodeReader, navigating/updating documents 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
XmlReader 2nd
XmlTextReader 2nd 3rd
XmlTextWriter 2nd 3rd 4th 5th 6th 7th 8th 9th
XmlValidatingReader 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
XmlWriter 2nd 3rd
combining with SQL (HttpHandlers) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
SqlHandler 2nd 3rd 4th 5th 6th 7th
SqlHandler output
XSL template used to format output 2nd 3rd 4th 5th
data
abbreviated close-tag syntax 2nd 3rd
CDATA elements 2nd 3rd

This document is created with the unregistered version of CHM2PDF Pilot

character entities 2nd 3rd 4th
elements that contain data 2nd 3rd 4th 5th
enclosing character data 2nd 3rd 4th 5th
navigating/updating documents with XmlNodeReader object 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
querying. [See XML, XPath queries]
using XmlDataReader object 2nd 3rd 4th 5th 6th 7th
viewing document data with XmlNode object 2nd 3rd 4th 5th 6th
writing with XmlTextWriter object 2nd 3rd 4th 5th 6th 7th
data access 2nd
Simple API for XML (SAX) 2nd 3rd
XML Document Object Model (DOM) 2nd 3rd 4th 5th 6th 7th 8th 9th
delimited data 2nd 3rd
enclosing character data 2nd 3rd 4th 5th
DISCO files
Document Object Model (DOM) 2nd 3rd 4th 5th 6th 7th 8th 9th
extended classes 2nd
fundamental classes 2nd 3rd
document structure/syntax
(begin)
(end)
abbreviated close-tag syntax 2nd 3rd
attributes 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th
CDATA elements 2nd 3rd
character entities 2nd 3rd 4th
declaration 2nd 3rd
elements (nodes) 2nd 3rd
elements containing data 2nd 3rd 4th
enclosing character data 2nd 3rd 4th 5th
Document Type Definitions (DTDs) 2nd 3rd 4th
elements (nodes)
ChildNodes class 2nd 3rd 4th 5th
complex/simple types 2nd 3rd 4th 5th 6th 7th
creating attributes 2nd
InsertAfter method 2nd 3rd
inserting new child nodes using AppendChild method 2nd
manipulating current nodes using XPathNodeIterator Current property 2nd 3rd
navigating/updating documents with XmlNodeReader object 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
viewing document data with XmlNode object 2nd 3rd 4th 5th 6th 7th 8th
expressing DataSet objects as XML 2nd 3rd 4th 5th
FirstChild() property 2nd
listings
10.1, comma-delimited document 2nd
10.10, books.xml document 2nd 3rd
10.11, loading a local XML file using XmlDocument .Load() method
10.12, loading an XML file residing on a Web server 2nd
10.13, loading a local XML file using XmlDocument .Load() method 2nd
10.14, ChildNodes class 2nd 3rd 4th
10.15, XmlTextReader object 2nd 3rd
10.16, XmlTextWriter object 2nd 3rd
10.17, XmlTextWriter object, document produced by XmlTextWriter object 2nd
10.18, XmlNodeReader object, creating from XmlDocument object 2nd
10.19, XmlNodeReader object, traversing a document 2nd

This document is created with the unregistered version of CHM2PDF Pilot

10.2 XML 1.0 declaration
10.20, XPathNavigator object Select method 2nd
10.21, XpathNodeIterator Current property 2nd 3rd
10.22, XmlNode object InsertAfter method 2nd 3rd
10.23, inserting new child nodes using AppendChild method 2nd
10.24, XPath query page used to test XPath queries against books.xml 2nd
10.25, XPath query results 2nd
10.26, XPath query to retrieve author information based on author ID 2nd
10.27, XPath query to retrieve multiple instances of same author 2nd 3rd
10.28, XPath query combining multiple criteria with AND and OR 2nd
10.29, XPath query that excludes nodes based on attribute values 2nd
10.3 XML document with a declaration and top-level element
10.3, XML document with a declaration and a top-level element
10.30, XPath query that retrieves a specific BOOK node by querying on text in BookÕs TITLE node 2nd
10.30, XPath query that retrieves a specific TITLE node by querying on its text 2nd
10.32, XDR Schema 2nd
10.33, simplified book.xml document 2nd
10.34, W3C schema definition boilerplate 2nd
10.35, W3C schema containing a definition for a simple type 2nd
10.36, W3C schema containing an element definition that refers to a type definition 2nd
10.37, W3C schema containing a complex type that refers to a simple type 2nd
10.38, Validation subroutine using XmlValidatingReader object 2nd 3rd
10.39, responding to events raised by Validate subroutine 2nd 3rd
10.4, XML document with elements that contain data 2nd
10.40, booksellerÕs hypothetical output
10.40, booksellerÕs hypothetical output
10.41, publisherÕs hypothetical input 2nd
10.42, XSLT Style Sheet That Converts an XML Document into an HTML Document 2nd 3rd
10.43, books.xml File Containing a Link to the books.xsl Style Sheet 2nd
10.44, programmatically transforming an XML file using an XSL style sheet and the XslTransform objec 2nd
10.5, XML document with multiple child elements beneath the top-level element 2nd 3rd
10.6, XML document with elements and attributes 2nd
10.7, XML document with a CDATA section 2nd
10.7, XML document with escape sequences 2nd
10.9, XML document with a empty elements 2nd
Load() method 2nd 3rd 4th 5th 6th
loading a local XML file 2nd 3rd
loading an XML file residing on a Web server
Microsoft XML Web site
navigating documents 2nd 3rd
schemas 2nd
case sensitivity
complex types 2nd 3rd 4th 5th 6th 7th
creating W3C XML schemas 2nd 3rd 4th 5th 6th 7th 8th
creating XSD schemas with Visual Studio .NET 2nd 3rd
Document Type Definitions (DTDs) 2nd 3rd 4th
listing 10.34, W3C schema definition boilerplate 2nd
listing 10.35, W3C schema containing a definition for a simple type 2nd
listing 10.36, W3C schema containing an element definition that refers to a type definition 2nd
listing 10.37, W3C schema containing a complex type that refers to a simple type 2nd
simple types 2nd 3rd 4th 5th 6th 7th
validating documents using W3C schemas 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th

This document is created with the unregistered version of CHM2PDF Pilot

XML Data-Reduced (XDR) schemas 2nd 3rd 4th 5th
SOAP (Simple Object Access Protocol) 2nd
style sheets, transforming documents 2nd 3rd 4th 5th 6th 7th
associating documents with style sheets 2nd 3rd
creating style sheets 2nd
creating XSLT style sheets 2nd 3rd 4th
editing XSLT files with Visual Studio .NET
programmatically performing XSL transformations 2nd 3rd
transforming documents using style sheets 2nd 3rd 4th 5th 6th 7th
associating documents with style sheets 2nd 3rd
creating style sheets 2nd
creating XSLT style sheets 2nd 3rd 4th
editing XSLT files with Visual Studio .NET
programmatically performing XSL transformations 2nd 3rd
updating documents using XmlNodeReader 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th
changing values 2nd 3rd 4th 5th 6th 7th
navigating documents 2nd 3rd
XPath queries. [See XML, XPath queries]
Web services. [See Web services]2nd [See Web services]
XML data representation versus HTML
HTML 2nd 3rd 4th 5th 6th 7th 8th 9th
XML 2nd 3rd
XPath queries 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th
combining multiple criteria with AND and OR 2nd
common XPath scenarios
dot (.) operator
excluding nodes based on attribute values 2nd
inequality operator (!=)
manipulating current nodes using XPathNodeIterator Current property 2nd 3rd
retrievig the root node of a document
retrieving a specific BOOK node by querying on text in BookÕs TITLE node 2nd
retrieving a specific TITLE node by querying on its text 2nd
retrieving author information based on author ID 2nd
retrieving multiple instances of same author 2nd 3rd
running a simple query 2nd
W3C XPath Web site
XPath query page used to test XPath queries against books.xml 2nd 3rd
XML Data-Reduced (XDR) schemas 2nd 3rd 4th 5th
XML Schema Definition Tool (xsd.exe)
XmlDataReader object 2nd 3rd 4th 5th 6th 7th
XmlDocument class 2nd 3rd 4th 5th 6th 7th 8th
XMLDocument class
creating XmlNodeReader objects from XmlDocument objects
XmlNamedNodeMap class 2nd
XmlNode class 2nd 3rd 4th
InsertAfter method 2nd 3rd
viewing document data 2nd 3rd 4th 5th 6th 7th 8th
XmlNodeList class 2nd
XmlNodeReader class 2nd 3rd 4th
navigating/updating documents 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th 20th
changing values 2nd 3rd 4th 5th 6th 7th

This document is created with the unregistered version of CHM2PDF Pilot

navigating documents 2nd 3rd
XPath queries. [See XML, XPath queries]
XmlReader class 2nd
XmlTextReader class 2nd 3rd
XmlTextWriter class 2nd
writing data 2nd 3rd 4th 5th 6th 7th
XmlValidatingReader class 2nd 3rd
validating schemas 2nd 3rd 4th 5th 6th 7th 8th
XmlWriter class 2nd 3rd
XPath queries (XML) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th

common XPath scenarios
dot (.) operator
inequality operator (!=)
listings
10.24, XPath query page used to test XPath queries against books.xml 2nd 3rd
10.25, running a simple query 2nd
10.26, retrieving author information based on author ID 2nd
10.27, retrieving multiple instances of same author 2nd 3rd
10.28, combining multiple criteria with AND and OR 2nd
10.29, excluding nodes based on attribute values 2nd
10.30, retrieving a specific TITLE node by querying on its text 2nd
10.31, retrieving a specific BOOK node by querying on text in BookÕs TITLE node 2nd
manipulating current nodes using XPathNodeIterator Current property 2nd 3rd
retrievig the root node of a document
W3C XPath Web site
XSD schemas
creating with Visual Studio .NET 2nd 3rd
xsd.exe (XML Schema Definition Tool)
xsdall element (W3C XML schemas) 2nd
xsdsequence element (W3C XML schemas)
xsdsimpleType node (W3C XML schemas)
XSLT files
editing with Visual Studio .NET
XSLT style sheets 2nd 3rd 4th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

Y switch (XCOPY) 2nd 3rd
Yes mode (XCOPY deployment 2nd 3rd 4th

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

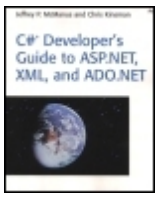
for RuBoard

Z switch (XCOPY)

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard



C# Developer's Guide to ASP.NET, XML, and ADO.NET

By Jeffrey P. McManus, Chris Kinsman

Publisher : Addison Wesley
Pub Date : March 29, 2002
ISBN : 0-672-32155-6
Pages : 608
Slots : 1

The book every Internet application developer working with Microsoft development tools needs to retool their knowledge of the new .NET techniques used to build Windows applications.

-
-
- Unbiased, in-depth commentary on the efficacy of the various technologies that comprise .NET as they pertain to Internet database developers.
-
- Technical know-how without crushing the reader with pointless detail.
-
- Implementation details that replace and extend the existing Active Server Pages (ASP), XML, and ActiveX Data Object (ADO) functionality currently supported by Microsoft.

Topics covered in this book include: the .NET Foundation Classes that are most used by developers--ASP.NET, XML, and ADO.NET, and details about the construction of Web Services and how they programmatically communicate with each other.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

About the Authors

Jeffrey P. McManus is a developer and speaker specializing in Microsoft tools. As a developer, he has specialized in online application development using Internet and client/server technologies. He is the author of four books on database and component technologies, including the bestselling *Database Access with Visual Basic 6* (Sams Publishing). Jeffrey regularly speaks at the VBITS/VSLive, European DevWeek, and VBConnections conferences.

Chris Kinsman is a developer and speaker specializing in Microsoft tools. As a developer, he has been responsible for several high-traffic sites based entirely on Microsoft tools, including serving as Vice President of Technology at DevX.com. In addition, Chris spent 10 years consulting with Fortune 500 companies throughout the world to solve their needs by utilizing a variety of Microsoft Visual Studio and Back Office technologies. Chris regularly speaks at the VBITS/VSLive, Web Builder, and SQL2TheMax conferences.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

About the Contributor

Anjani Chittajallu obtained a masters degree from Indian Institute of Technology (I.I.T-Madras) with a major in Control Systems Engineering. She specializes in designing and developing enterprise systems with Microsoft Technology. Anjani currently holds MCSD certification. She can be reached at srianjani@hotmail.com.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

About the Technical Editor

Joel Mueller is a senior software engineer at DeLani Technologies (www.delani.com), a leading Web development software company, where he has been spearheading the company's Microsoft .NET development effort since July 2000. Prior to the advent of ASP.NET, Joel did extensive work with Microsoft Active Server Pages and Macromedia ColdFusion. He has written for several books and articles on the topics of Macromedia ColdFusion and XML. Joel's current interests include the .NET framework, C#, and sleeping.

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard

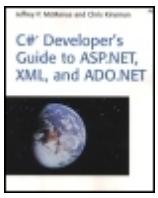
Acknowledgments

Jeffrey and Chris would like to extend a special thank you to Anjani Chittajallu, who came through in a pinch and did a bang-up job working on the code examples for this edition of the book. We're grateful to you for your help!

for RuBoard

This document is created with the unregistered version of CHM2PDF Pilot

for RuBoard



C# Developer's Guide to ASP.NET, XML, and ADO.NET

By Jeffrey P. McManus, Chris Kinsman

Publisher : Addison Wesley
Pub Date : March 29, 2002
ISBN : 0-672-32155-6
Pages : 608
Slots : 1

Copyright
About the Authors
About the Contributor
About the Technical Editor
Acknowledgments
Chapter 1. Introduction: The Need for ASP.NET
- Problems with ASP Today
- Introducing ASP.NET

Chapter 2. Page Framework
- ASP.NET's Control Model
- Separating Presentation from Code Using Code Behind
- Programming HTML Controls
- Attributes of the Page Object
- Creating User Interfaces with Web Controls
- Server Controls and Page Object Reference

Chapter 3. Debugging ASP.NET Applications
- Tracing Your Web Application's Activity
- Debugging ASP.NET Applications
- Creating Custom Performance Monitors
- Writing to the Windows Event Log
- Reference

Chapter 4. State Management and Caching
- State Management: What's the Big Deal?
- Caching
- Class Reference

Chapter 5. Configuration and Deployment
- Understanding Configuration Files
- Global and Local Configuration Files
- Structure of Configuration Files
- Accessing Configuration Files Programmatically
- Editing Web Configuration Files in Visual Studio .NET
- Initializing Web Applications Using Global.asax

This document is created with the unregistered version of CHM2PDF Pilot

Using XCOPY for Deployment
Managing the Global Assembly Cache

Chapter 6. Web Services
Historical Influences
Network Data Representation
What Is a Web Service?
Why Web Services?
ASP.NET Web Services
Consuming Web Services
Class Reference

Chapter 7. Security
Identity and Principal
Windows Authentication
Forms Authentication
Passport Authentication
File Authorization
URL Authorization
Custom Roles with Forms Authentication
Pulling It All Together
Impersonation
Class Reference

Chapter 8. HttpHandlers and HttpModules
An Overview of ASP.NET Request Handling
HttpModules
HttpHandlers
Dynamic Handler Assignment
Class Reference

Chapter 9. Building User Controls and Server Controls
Working with User Controls in Web Forms Applications
Creating Server Controls

Chapter 10. Using XML
What Is XML?
Accessing XML Data Using .NET Framework Classes
Defining and Validating XML with Schemas
Processing XML Documents Using Style Sheets
Class Reference

Chapter 11. Creating Database Applications with ADO.NET
Why a New Object Library for Data Access?
New Features in ADO.NET
Connecting to a Database
Running Queries
Using Data Adapters to Retrieve and Manipulate Data
Creating Web Forms for Data Entry
Handling Errors
ADO.NET Framework Reference

This document is created with the unregistered version of CHM2PDF Pilot

Index
for RuBoard